TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JOONAS IHONEN

KESKEYTYSPOHJAINEN PULSSILASKURI SULAUTETUSSA LINUX-JÄRJESTELMÄSSÄ

Diplomityö

Tarkastaja: professori Timo D. Hämäläinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
07. marraskuuta 2012

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Sähkötekniikan koulutusohjelma
IHONEN, JOONAS: Keskeytyspohjainen pulssilaskuri sulautetussa Linux-järjestelmässä
Diplomityö, 55 sivua, 7 liitesivua
Kesäkuu 2013
Pääaine: Ohjelmoitavat alustat ja laitteet
Tarkastaja: professori Timo D. Hämäläinen
Avainsanat: sulautettu järjestelmä, käyttöjärjestelmä, Linux, ajuri, keskeytys, reaaliaika, pulssilaskuri

Linux on viime vuosina saavuttanut suuren suosion sulautettujen järjestelmien käyttöjärjestelmänä. Alunperin PC-käyttöön suunniteltu Linux-ydin ei sellaisenaan tarjoa kovin kattavia reaaliaikaisuusominaisuuksia, mutta erinäisiä reaaliaikapäivityksiä on saatavilla. Tässä työssä on tarkasteltu Linux-ydintä, johon on asennettu RT Patch -niminen reaaliaikapäivitys.

Työn päämääränä oli toteuttaa pulssilaskuri sulautettuun Linux-järjestelmään. Tässä työssä pulssilaskurilla tarkoitetaan laskuria, joka laskee digitaalisen signaalin muutosten määrää, aikaväliä ja taajuutta. Pulssilaskuria sovellettiin käytännössä kierrosnopeuden mittaamiseen.

Pulssilaskuri toteutettiin Linuxin laiteajurina. Laiteajurin suunnittelu Linuxille vaatii hyvää laitteiston ja käyttöjärjestelmän tuntemista. Keskeytyksiä käyttävä laiteajuri ei saa aiheuttaa ongelmia rinnakkaisuudessa eikä reaaliaikaisuudessa.

Pulssilaskuri mittaa tapahtumia asynkronisesti, mutta tavallisesti siltä luetaan arvoja synkronisesti. Tällöin keskiarvon laskentamenetelmään on kiinnitettävä huomiota hyvän tarkkuuden saavuttamiseksi. Työssä tarkasteltiin keskiarvon oikeellisuutta kahdella eri keskiarvomenetelmällä.

Valmiin laiteajurin toimintaa varmennettiin mittaamalla reaaliaikaisuutta ja tarkkuutta. Reaaliaikaisuutta arvioitiin mittaamalla latenssia ja keskeytyskäsittelijän pituutta. Tarkkuutta arvioitiin vertaamalla laiteajurin laskemia arvoja signaalinlähteenä toimivan funktiogeneraattoriin asetettuihin arvoihin.

Laiteajurin testeissä mitattiin 1000Hz testisignaalilla keskihajonnaksi 0,54Hz ja poikkeamaksi 2,9Hz. Samalla taajuudella mitattiin latenssin keskiarvoksi 25µs ja huonoimman tapauksen arvoksi 85µs. Suurimmaksi toimintataajuudeksi määritettiin kokeellisesti 85kHz.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Electrical Engineering
IHONEN, JOONAS: Interrupt-based Pulse Counter on Embedded Linux System
Master of Science Thesis, 55 pages, 7 Appendix pages
June 2013
Major: Programmable Platforms and Devices
Examiner: Professor Timo D. Hämäläinen
Keywords: embedded system, operating system, Linux, driver, interrupt, real-time, pulse counter

In recent years, Linux has achieved great popularity in embedded systems. Being originally designed for PC use, the Linux kernel does not provide very comprehensive real-time features. Currently there is a variety of real-time patches available. This thesis deals with the Linux kernel 2.6 with RT Patch.

The goal was to implement a pulse counter on an embedded Linux system. In the context of this work, a pulse counter is defined as a counter that counts a number of digital signal changes and calculates interval and frequency. The pulse counter was used in a tachometer application that measures rotational speed.

Pulse counter was implemented as a Linux device driver. Device driver design requires good knowledge of hardware and operating system. A driver that uses interrupts is not allowed to cause concurrency or real time issues.

Pulse counter measures events asynchronously, but is usually polled synchronously. Therefore average calculation method needs attention to reach good accuracy. Correctness of average was examined with two different averaging methods.

Implemented driver was verified by measuring real-time responsibility and accuracy. Real-time responsibility was assessed by measuring interrupt latency and time usage of the interrupt service routine. Accuracy was estimated by comparing values calculated by the driver to values of the function generator

Measurements with 1000 Hz test signal resulted in 0,54 Hz of statistical dispersion and 2,9 Hz of deviation. Latency average was 25 µs and the worst case latency was 85 µs. Maximum operation frequency was defined experimentally to 85 kHz.

ALKUSANAT

Tämä diplomityö on tehty Wapice Oy:lle vuosien 2012 ja 2013 aikana. Työn tavoitteena oli tehdä laiteajuri pulssilaskurille sulautettuun Linux-järjestelmään yrityksen omaan tuotteeseen, ja tutkia samalla järjestelmän reaaliaikaisuutta.

Haluan kiittää työn tarkastajaa professori Timo D. Hämäläistä ja ohjaajaa Tom Hanneliusta ohjauksesta ja neuvoista. Lisäksi haluan kiittää Risto Pajulaa työn katselmoinnista ja teknisistä neuvoista sekä Teemu Siuruaista ja Jouko Haapaluomaa Linux-asiantuntemuksen jakamisesta. Wapice Oy:n työyhteisö on ollut merkittävä tuki ja tiedon lähde työtä tehdessä. Ennen kaikkea haluan kiittää vaimoani Johannaa, joka on ollut tukenani koko tämän diplomityön ajan.

Tampereella 15.5. 2013

Joonas Ihonen

SISÄLLYS

LYHENTEET JA MERKINNÄT

AIC	Advanced Interrupt Controller, keskeytysohjain
API	Application programming interface, oihjelmointirajapinta
ARM	Advanced RISC Machines, Acorn Computersin kehittämä 32-bittinen suoritinarkkitehtuuri
AT91SAM9260	Atmelin valmistama suoritinyksikkö ARM926-suorittimella
CAN	Controlled area network, automaatioväylä
FIQ	Fast interrupt request, korkean prioriteetin keskeytys ARM-arkkitehtuurissa
GSM	Global System for Mobile Communications, mobiilikommunikaatiopalvelu
GPRS	General Packet Radio Service, mobiilikommunikaatiopalvelu
GPS	Glopal Positioning System, satelliittipaikannusjärjestelmä
I/O	Input/output, sisään- ja ulostulo
ISR	Interrupt service routine, keskeytyskäsittelyn suorittava aliohjelma
IRQ	Interrupt request, keskeytypyyntö
MCK	Master Clock, ARM-suorittimen pääkello
MPPC	Multi-Purpose Pulse Cousnter, työssä toteutettu Linux-laiteajuri pulssilaskentaan
MPU	Microprocessor unit, suoritinyksikkö
ROM	Read-only Memory, muisti johon voidaan vain kirjoittaa
MPU	Microprocessor unit, suoritinyksikkö
RAM	Random-access memory, muistityyppi
RS-485	Sarjaliikennestarndardi
RTOS	Real-time operating system, reaaliaikakäyttöjärjestelmä
SRAM	Static random-access memory, muistityyppi
VFS	Virtual filesystem, virtuaalinen tiedostojärjestelmä
VRS	Variable reluctance sensor, etäisyysanturityyppi
WRM	Wapice Remote Management, Wapice Oy:n etähallintajärjestelmä

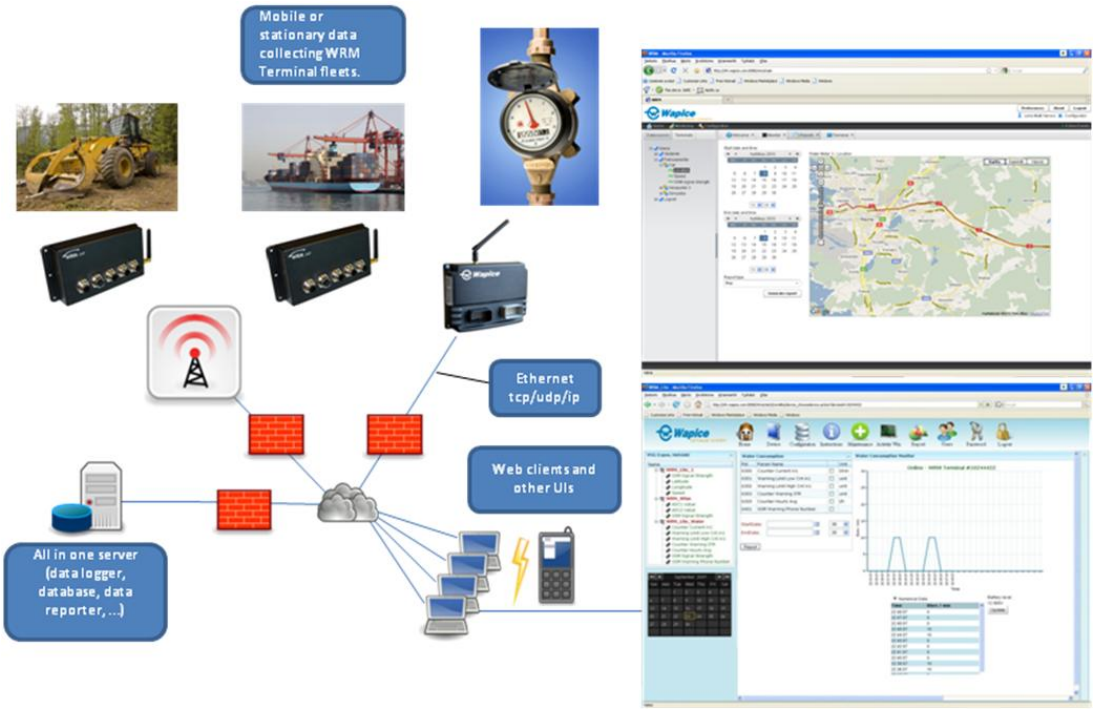
1 JOHDANTO

1.1 Tausta

Wapice Oy on vuonna 1999 Vaasassa perustettu informaatioteknologiayritys, joka työllistää tällä hetkellä noin 230 henkilöä. Yritys tekee pääosin ohjelmistoa ja elektroniikkaa alihankintana, mutta sillä on myös omia tuotteita, kuten Wapice Remote Management ja Summium.

Wapice Remote Management eli WRM on Wapicen kehittämä etähallintajärjestelmä. Se on suunniteltu monenlaiseen etähallintaan ja tiedonkeräämiseen langallisesti tai langattomasti. Järjestelmä kattaa laitteiston, palvelimen, ohjelmiston ja käyttöliittymän. Järjestelmästä on pyritty tekemään yleiskäyttöinen, mutta sitä myös muokataan tilaajan tarpeisiin sopivaksi. WRM:n käyttökohteita ovat muun muassa ajoneuvojen seuranta, automaatiolaitteiden etähallinta ja vedenkulutuksen seuranta.

Järjestelmän pääosat ovat palvelin ja terminaalit. Terminaalit keräävät ja lähettävät tietoa palvelimelle, jossa sitä voidaan tarkastella ja käsitellä. Tieto voi olla esimerkiksi GPS-paikannustietoa, CAN-dataa tai analogisen lämpötila-anturin arvoja. Palvelimen käyttöliittymän avulla käyttäjä voi hallita terminaaleja ja päivittää niiden ohjelmistoa. Terminaalit sisältävät useita liitäntämahdollisuuksia, kuten analogisia ja digitaalisia I/O-portteja sekä erinäisiä väyliä, kuten CAN ja RS-485. Kuvassa 1.1 on esimerkki WRM-järjestelmän kokoonpanosta.



Kuva 1.1. *Wapice Remote Management -järjestemän periaatekuva*

Terminaalilaitteista on tarjolla eri vaihtoehtoja. Tämä diplomityö käsittelee WRM 247-terminaalia, joka on tällä hetkellä yleisimmin käytetty WRM:n terminaalilaite. Toinen Wapicen valmistama terminaali on monipuolisemmilla liitännöillä ja laajemmalla jännitealueella varustettu WRM 365. Lisäksi WRM-ohjelmistoa on käytetty muiden valmistajien kaupallisilla laitteilla kuten teollisuus-PC:llä. WRM 247 on sulautettu järjestelmä, jonka käyttöjärjestelmänä on Linux.

Työn toisena osapuolena on Moventas Gears Oy, joka on vuonna 2005 perustettu tuuliturbiinivaihteita ja muita teollisuusvaihteita valmistava yhtiö. Se työllistää tällä hetkellä noin 1000 työntekijää 14 maassa. Moventas Gears toimii työn osittaisena tilaajana.

Moventas Gears:n tilaama projekti on osana valvontajärjestelmää, jolla tarkkaillaan esimerkiksi tuuliturbiinin vaihteen kierrosnopeutta ja värähtelyä. Tähän otettiin koekäyttöön WRM 247, johon tehtiin oma ohjelmisto projektia varten. PC:lle tehtiin graafisella käyttöliittymällä varustettu ohjelma, johon terminaalilaite lähettää tiedot. Toteutus eroaa sikäli WRM:sta, että varsinaista palvelinta ei ole käytössä. PC-sovelluksen osuus on jätetty tämän diplomityön tarkastelualueen ulkopuolelle.

1.2 Tavoitteet

Tutkimuksen tavoitteena on toteuttaa Wapicen WRM 247 -terminaalilaitteeseen ohjelmisto, joka mittaa pulssimuotoisen signaalin taajuutta. Mittaus on kyettävä suorittamaan riittävällä tarkkuudella ja riittävän suurella taajuudella. Taajuus muutetaan kierrosnopeudeksi ja sen arvo välitetään PC-sovellukselle, jossa se näytetään kokonaislukuna. Moventas Gears:n vaatimus on, että ohjelma pystyy mittaamaan taajuuksia väliltä 30-1000Hz. Tarkkuuden vaatimuksena on 100rpm kierrosnopeudella ±1rpm ja sitä suuremmilla kierrosnopeuksilla ±1%. Tämä vaatimus on asetettu 2 sekunnin mittausten keskiarvolle.

Kun Moventas Gears:n asettamat toiminnalliset vaatimukset saadaan täytettyä, on tarkoitus kehittää kierrosnopeuslaskurista yleiskäyttöisempi pulssilaskuri, joka voidaan asettaa mittaamaan myös pulssien lukumäärää ja pulssin pituutta. Työn tavoitteena on myös toteutuksen ohella tutkia WRM 247:n ja Linux-ytimen 2.6 reaaliaikaisuutta. Tätä tietoa voidaan hyödyntää esimerkiksi tulevia laiteajureita tehtäessä tai mahdollisesti uutta laitteistoa suunnitellessa.

1.3 Tutkimuksen rakenne

Aluksi luvussa 2 selvitetään, mitä pulssilaskennalla tarkoitetaan tämän työn yhteydessä, ja esitellään pulssilaskennan sovelluksia. Tämän jälkeen luvussa 3 tutustutaan työssä käytettyyn laitteistoon tärkeimmiltä osiltaan. Laitteiston pääosat ovat suoritin oheislaitteineen, anturit ja sisääntulokytkentä.

Luvussa 4 perehdytään ohjelmistoa käsittelevään teoriaan. Tämä osuus keskittyy lähinnä Linux-käyttöjärjestelmän ytimen kuvaamiseen eri tasoilla. Tässä yhteydessä puhutaan myös reaaliaikaisuudesta, joka on yleisesti ottaen huomioonotettava seikka sulautetuissa järjestelmissä.

Luvussa 5 pohditaan seikkoja, jotka pitää ottaa huomioon työtä toteuttaessa. Tässä sivutaan niin matemaattisia kysymyksiä kuin ohjelmistoon liittyviä asioita. Luku 6 kuvailee työssä toteutettua ohjelmistoa.

Luvuissa 7 ja 8 käsitellään mittaustuloksia. Näistä edellinen keskittyy reaaliaikaisuutta tarkasteleviin mittauksiin ja jälkimmäinen tarkkuutta käsitteleviin mittauksiin. Lopuksi luvussa 9 on yhteenveto koko työstä.

2 PULSSILASKENTA

2.1 Pulssilaskennan teoriaa

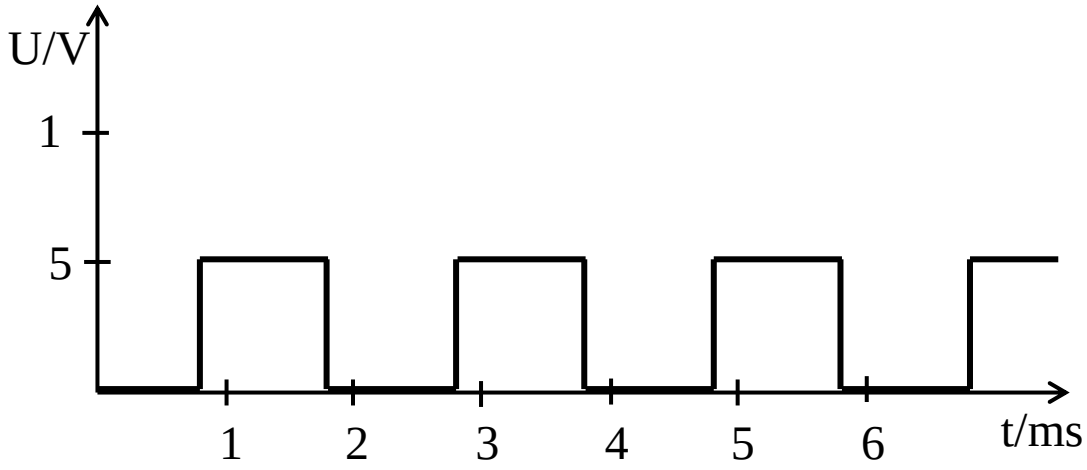
Pulssi tarkoittaa tavallisesti digitaalitekniikassa digitaalisen signaalin muutosta tilasta toiseen ja takaisin. Tavallisen pulssilaskurin käyttötarkoitus on laskea näiden pulssien määrää. Käytännössä tämä tehdään yleensä kasvattamalla laskurin arvoa, kun havaitaan uusi samansuuntainen pulssinreuna. Voidaan tarkastella joko nousevia tai laskevia reunoja. Voidaan myös laskea pulssin jokainen reuna, jolloin taajuus on kaksinkertainen verrattuna edellä mainittuun.

Pulssista voidaan mitata sovelluksesta riippuen myös esimerkiksi pulssin pituutta eli jaksonaikaa, taajuutta, ylhäällä- tai alhaallaoloaikaa tai pulssisuhdetta. Pulssin pituus T saadaan mittaamalla kahden peräkkäisen samansuuntaisen pulssinreunan aikaero kaavalla

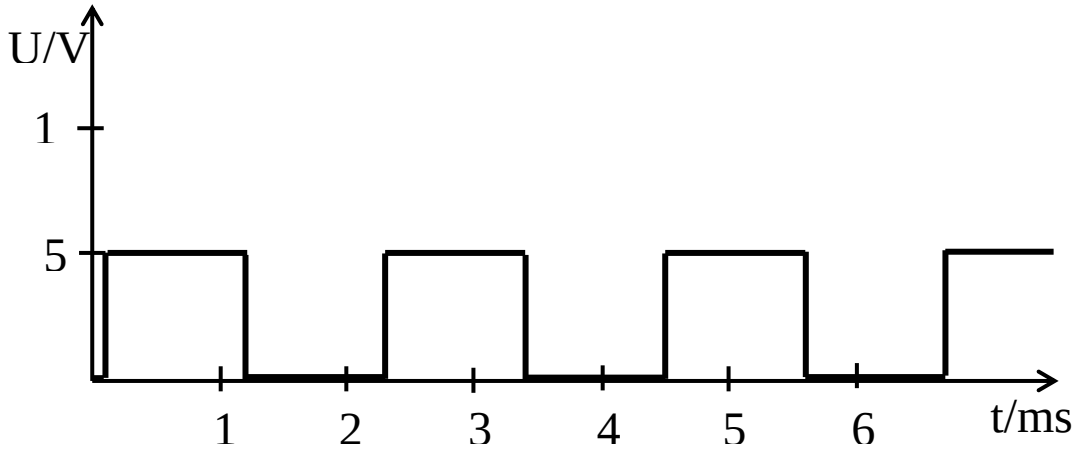
$$T = t_n - t_{n-1}, \tag{2-1}$$

missä t_n on pulssinreunan n ajanhetki. Taajuus saadaan laskemalla pulssien määrä aikayksikköä kohden tai laskemalla pulssinpituuden käänteisluku. Peräkkäisten nousevan ja laskevan pulssinreunan välisestä aikaerosta saadaan joko ylhäälläoloaika tai alhaallaoloaika siten, että jos nouseva reuna on tapahtunut ensin, on kyseessä ylhäälläoloaika, ja jos laskeva reuna on tapahtunut ensin, on kyseessä alhaallaoloaika.

Muutamia asioita on hyvä ottaa huomioon taajuusmittausta suunnitellessa. Jos mitataan pulssien määrää aikayksikköä kohden, on mittaustarkkuus riippuvainen mittausvälin pituudesta. Seuraava esimerkki havainnollistaa ilmiötä. Kuva 2.1 esittää 500Hz kanttiaallon mittausta. Siinä on kuuden millisekunnin aikavälillä kolme nousevaa reunaa. Tästä saadaan pulssien taajuudeksi 3/(0,006s) = 500Hz. Kuvassa 2.2 on 455Hz kanttiaaltoa. Siinä on myös kuuden millisekunnin aikavälillä yhä kolme signaalin muutosta, joten mitatuksi taajuudeksi saadaan yhä 500Hz. Virhe on tällöin noin 10%. Tarkkojen tulosten saamiseksi mittausvälin pitää olla riittävän pitkä suhteessa pulssin pituuteen. Mittaustavan valinta on syytä tehdä käyttökohteen perusteella. Taajuudesta voidaan laskea myös useamman mittausnäytteen keskiarvo, jolloin edellä mainitun kaltaisten virhetekijöiden merkitys pienenee.



Kuva 2.1. 500Hz kanttiaaltosignaali



Kuva 2.2. 455Hz kanttiaaltosignaali

Työssä käytetty laitteisto ei juurikaan rajoita sitä, kuinka korkeita taajuuksia voidaan mitata. Kappaleessa 3.2 esiteltävän AT91SAM9260-suoritinyksikön I/O-linjat kykenevät havaitsemaan pulsseja 50MHz maksimitaajuudella 1,8V käyttöjännitteellä [3]. Tämä on määritelty pulssisuhteella 40-60%. Rajoitukset suurimpaan mitattavissa olevaan taajuuteen tulevat ohjelmiston suorituskyvystä.

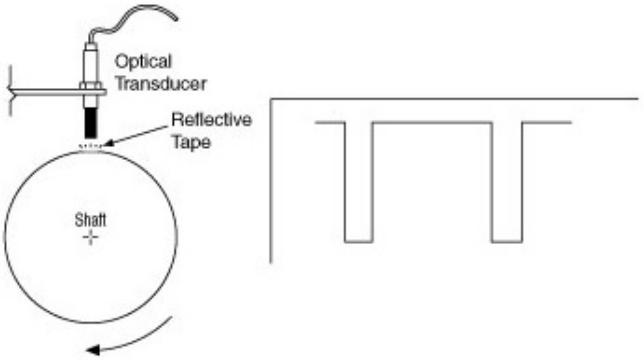
2.2 Kierrosnopeusmittaus

Tavallisesti kierrosnopeutta mitataan etäisyysanturilla, jotka ovat yleensä sähkömagneettisia tai fotosähköisiä antureita. Sähkömagneettisia antureita ovat muun muassa Hall-anturi, induktiiviset anturit ja magnetoresistiiviset anturit. Sähkömagneettinen kierrosnopeusanturi koostuu kahdesta osasta. Toinen osa on pyörivässä roottorissa, ja se koostuu joko yhdestä tai useammasta kestomagneetista tai ferromagneettisesta hammaspyörästä. Toinen osa on paikallaan olevan staattorin puolella, ja se on joko Hall-anturi tai induktiivinen anturi. Osat on mekaanisesti erotettu, ja ne on sijoitettu yleensä muutaman millimetrin päähän toisistaan. Sähkömagneettisen anturin käyttö on yleensä halpaa, ja installaatio on varsin luotettava. Haittapuolena erottelukyky on alhainen ja mag-

neettiset osat pitää erottaa siten, että ne eivät häiritse järjestelmän toimintaa. Toteutustapa soveltuu pitkän käyttöiän kohteisiin, joita ovat esimerkiksi autot, tehdasautomaatiolaitteet ja generaattorit. [7]

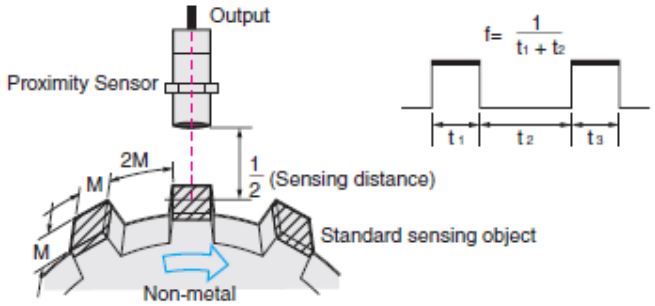
Fotosähköiset anturit koostuvat myös kahdesta osasta, joista toinen on roottorin ja toinen staattorin puolella. Toteutustapoja on monia. Niistä eräs on muovinen kiekko, jonka kehällä on tiheään vuoroin läpinäkyviä ja läpinäkymättömiä osia. Läpinäkyvät osat voivat olla esimerkiksi kiekkoon porattuja reikiä, tai kiekko on voitu valmistaa optisella litografialla. Staattorin puolella on valonlähde, joka on tavallisesti LED tai LASER, ja valoherkkä anturi, joka on hyvin lähellä roottorin muovikiekkoa. Jos muovikiekon läpinäkymättömät osat ovat tiheässä, pystyy fotosähköinen anturi tarjoamaan hyvinkin suuren erottelukyvyn. Ne ovat tavallisesti sähkömagneettisia antureita kalliimpia, ja niitä käytetään sovelluksissa, joissa tarvitaan hyvin tarkkaa kierrosnopeusmittausta. Sovelluksia ovat esimerkiksi teollisuusrobotit, automaattinen tuotantolinjat, lentokoneteollisuuden tarpeet. Fotosähköiset anturit ovat herkkiä likaantumiselle ja tärinälle, mistä syystä esimerkiksi auton moottorissa niiden käyttö on ongelmallista. [7]

Toisenlainen tapa käyttää fotosähköistä anturia on sijoittaa anturi ja valonlähde samaan anturiyksikköön. Tällöin roottorin puolelle sijoitetaan yksi tai useampi heijastintarra, jolloin tarran ollessa anturiyksikön kohdalla, sen lähettämää valoa heijastuu takaisin anturille, minkä seurauksena anturi tuottaa pulssin. Kuvassa 2.3 on esimerkki fotosähköisen anturin installaatiosta, jossa on yksi heijastinteippi asennettu roottorille (kuvassa shaft).



Kuva 2.3. *Fotosähköisen anturin installaatio [31]*

Auton moottorin kierrosnopeutta mitataan yleensä sähkömagneettisella anturilla. Yleisimmät anturit ovat Hall-anturi ja VRS (variable reluctance sensor). Kierrosnopeutta mitatessa moottorin pääakseliin on kiinnitetty metallinen hammaspyörä, jota kutsutaan usein nimellä liipaisinpyörä (trigger wheel). Anturi sijoitetaan tämän hammaspyörän yhteyteen esimerkiksi kuvan 2.4 esittämällä tavalla. Kun hammas ohittaa anturin, syntyy pulssi. Kierrosnopeus saadaan laskemalla missä ajassa havaitaan yhtä monta pulssia kuin hammaskehässä on hampaita. Tämän käänteisarvona saadaan moottorin kierrosnopeus. [8]



Kuva 2.4. *Kierrosnopeuden mittaus liipaisinpyörän avulla [25]*

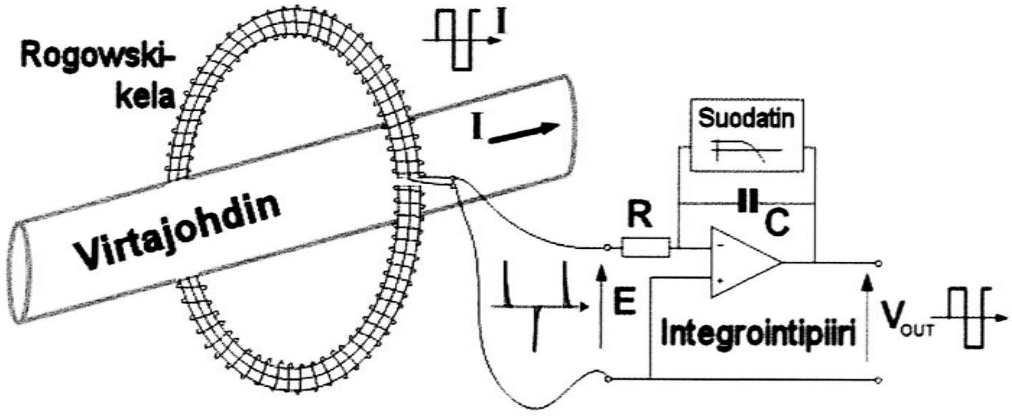
VRS koostuu kelasta, joka on käämitty kestomagneetin ympärille. Kun hammas ohittaa anturin, se synnyttää kelaan muuttuvan magneettikentän, mikä indusoi sähkövirran. Sähkövirta on positiivinen, kun hammas lähestyy anturia ja negatiivinen, kun hammas etääntyy anturista. Hall-anturissa anturielementti on magneetin ja kohteen välissä. Hall-anturi mittaa magneettikenttää, kun taas VRS mittaa sen muutosta. Anturin virtasignaali muutetaan elektroniikalla digitaaliseksi pulssisignaaliksi. Hall-anturia käytettäessä tämä saatetaan tehdä anturin välittömässä läheisyydessä. [8]

Jos kyseessä on auton polttomoottori, suurin kierrosnopeus on normaalisti alle 10000 kierrosta minuutissa, joka on noin 170 kierrosta sekunnissa. Esimerkiksi Saabin B234-moottorissa on VRS ja 62-hampainen hammaskehä [11]. Kaksi hammasta puuttuu kehästä, joten laskennallisesti hampaita on 64. Moottorin maksimikierrosnopeus on ohjelmallisesti rajoitettu noin 6200 kierrokseen minuutissa. Tällöin pulsseja tulee maksimissaan noin 6400 sekunnissa, eli pulsseja tulisi noin 15 mikrosekunnin välein.

2.3 Muita pulssimittauksen sovelluksia

Nesteen virtaus on yksi pulssimittauksen sovelluskohteita. Nesteen virtausta mitataan tavallisesti massana tai tilavuutena aikayksikköä kohden. Pulssimuotoisessa mittauksessa yksi pulssi ilmaisee tietyn nestetilaavuuden tai -massan virtaamista anturin läpi. Virtausmittareita on erilaisiin teknologioihin perustuvia. Hall-anturiin perustuva mittaus on yksi tapa. Toisessa menetelmässä nesteen virtaus pyörittää turbiinia, joka pyöriessään synnyttää pulsseja tietyn määrän kierrosta kohden. Käytössä on myös ultraääniantureita, joiden toiminta perustuu doppler-ilmiöön. Äänen kulkemisaika nesteessä mitataan kahden pisteen välillä. Nopeammalla virtauksella ääni siirtyy myös nopeammin. [10]

Sähköenergiankulutusmittarit käyttävät usein pulssimuotoista ulostulosignaalia. Yksi pulssi ilmaisee tietyn sähköenergiamäärän siirtymistä mittauspisteen läpi. Yksi pulssi voi esimerkiksi merkitä kilowatin energiaa. Käytännössä usein mitataan virtaa, mistä saadaan laskettua teho, kun tunnetaan jännite. Tehoa integroimalla saadaan laskettua johtimen kautta kulkenut sähköenergia. Kuvassa 2.5 on esimerkki eräästä virtamittausmenetelmästä. Tässä Rogowski-kelaan indusoituu jännite, jonka suuruus on verrannollinen johtimen virran muutosnopeuteen. Integrointipiirin avulla saadaan ulostuloon jännite, jonka suuruus on verrannollinen johtimen virtaan. Jännitesignaali voidaan muuntaa pulssisignaaliksi esimerkiksi sopivalla AD-muuntimella. [30]

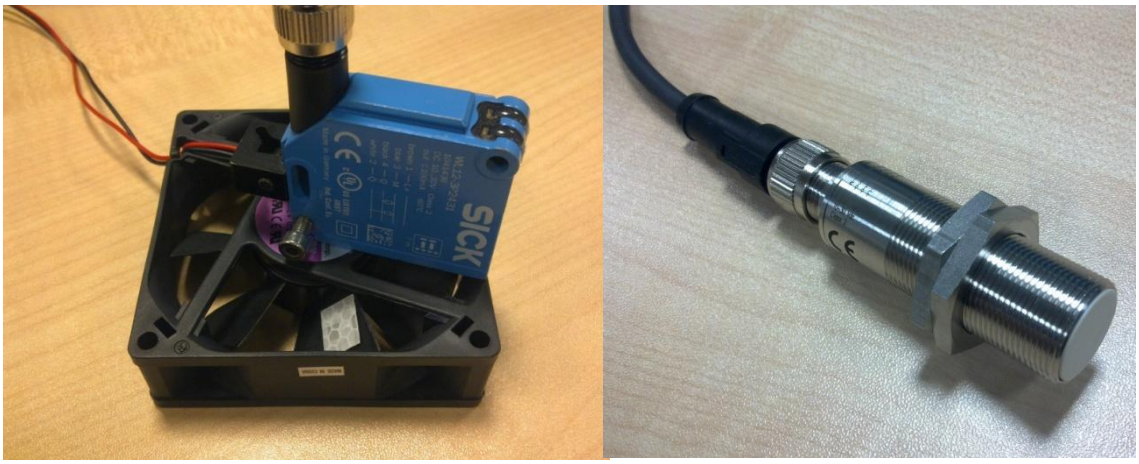


Kuva 2.5. *Periaatekuva virtamittauksesta [30]*

3 LAITTEISTO

3.1 Anturit

Työssä käytettiin signaalinlähteenä kahta erilaista etäisyysanturia. Toinen on fotosähköinen anturi mallia SICK W12-3P2431 [34] ja toinen induktiivinen anturi mallia SICK IMF18-08BPPVCOS [16].

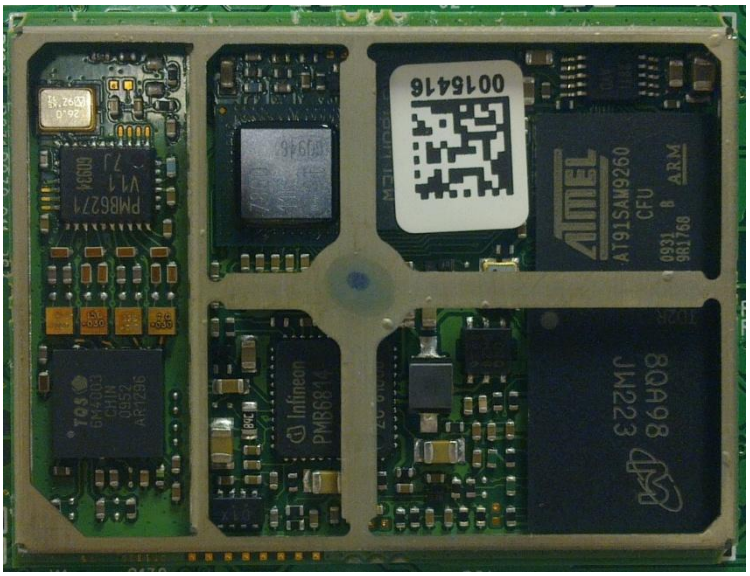


Kuva 3.1. *Fotosähköisen anturi testikokoonpanossa ja induktiivinen anturi*

Fotosähköisessä anturissa on samassa kotelossa myös valonlähde. Kun valo heijastuu takaisin anturille, se saa aikaan sähköisen pulssin. Testeissä anturi asetettiin PC:n kotelotuulettimen yhteyteen ja tuulettimen lapaan liimattiin heijastinteippi kuvan 3.1 mukaisesti. Induktiivista anturia ei käytetty todellisen käyttökohteen tapaan hammaskehän kanssa, vaan sen toiminta WRM 247:n kanssa todettiin liikuttelemalla metallikappaletta sen läheisyydessä.

3.2 Järjestelmäpiiri

WRM 247 -laite on rakennettu Telit Communications PLC:n valmistaman GE863-PRO^3-järjestelmäpiirin ympärille. Piirin pääosat ovat Atmel AT91SAM9260 suoritinyksikkö, GSM/GPRS-moduuli ja muistit. GSM/GPRS-moduuli on Telitin valmistama. Muistina on 128Mt NAND flash -muistia ja 64Mt SDRAM-muistia. Piiri on pakattu metallikoteloon, joka näkyy kuvassa 3.2 ilman kantta. [13]



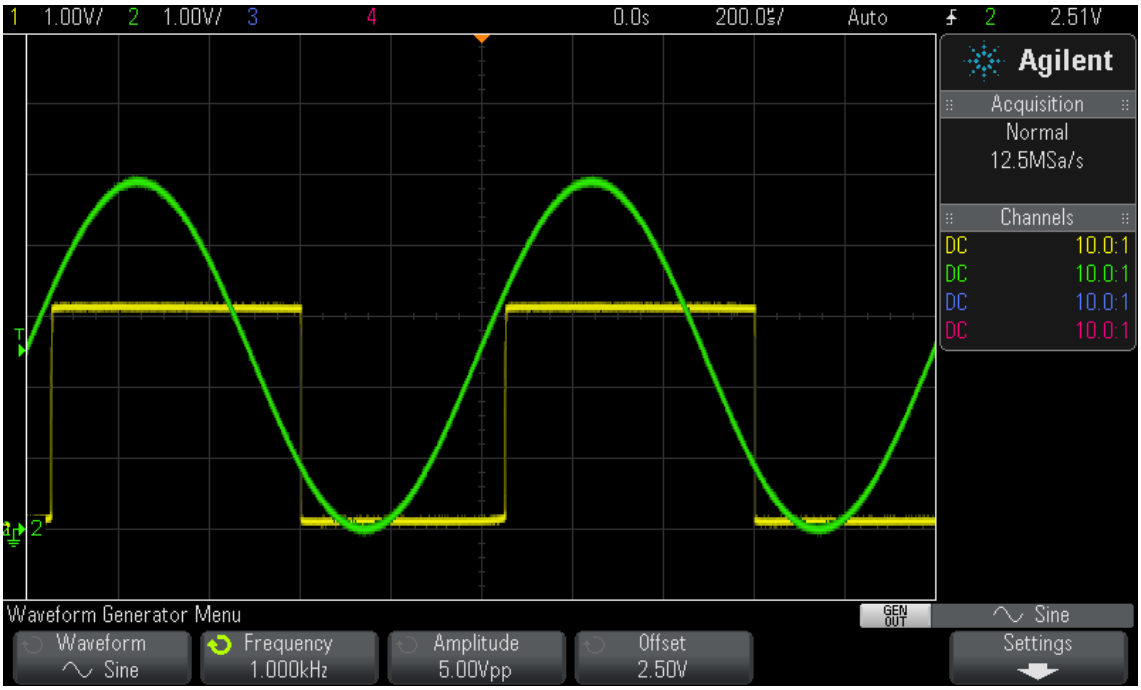
Kuva 3.2. *GE863-PRO3-järjestelmäpiiri*

AT91SAM9260 on ARM926EJ-S-suorittimella varustettu 32-bittinen suoritinyksikkö (microprocessor unit, MPU), joka toimii 200MHz kellotaajuudella. Se sisältää 8 kilotavua SRAM-muistia ja 32 kilotavua ROM-muistia. MPU sisältää suorittimen lisäksi monia muita ominaisuuksia, joista tämän työn kannalta merkittävimmät ovat keskeytysohjain, kaksi kolmikanavaista 32-bittistä ajastinta (timer/counter) ja 96 I/O-linjaa. [3]

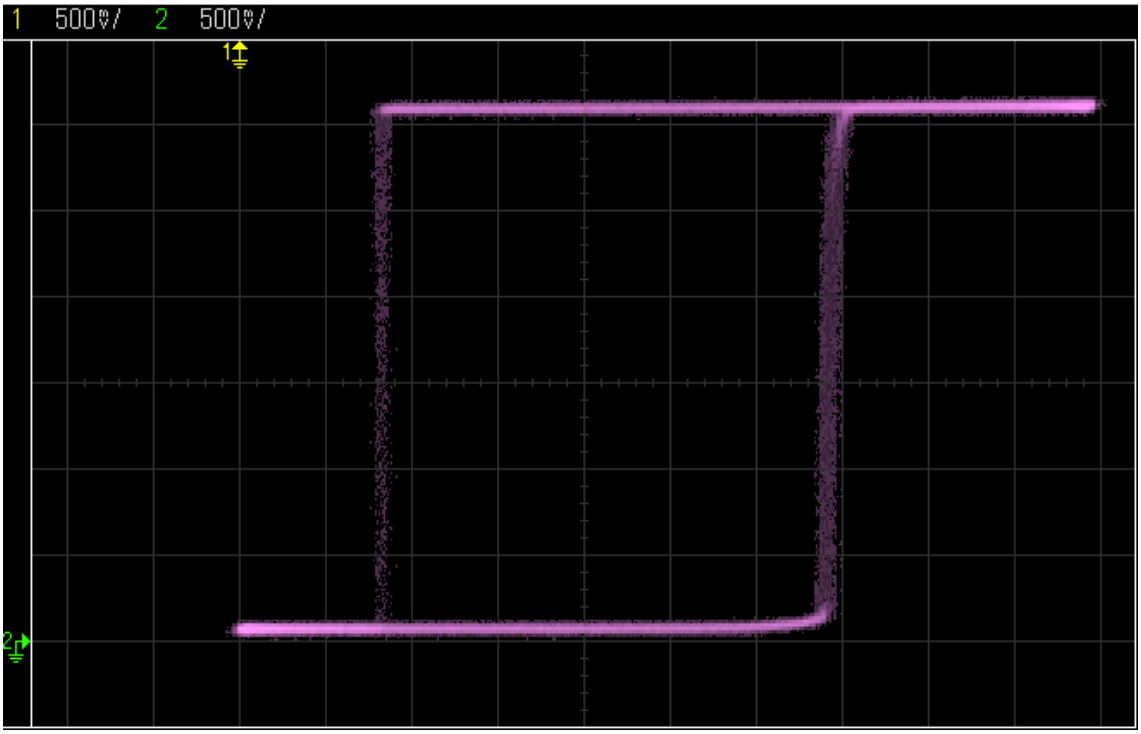
3.3 Sisääntulokytkentä

WRM 247:n digitaalisten sisääntulojen kytkentäkaavio on esitetty liitteessä E. Kaikki neljä sisääntulokytkentää ovat identtiset. Sisääntulossa on ferriitti R191 suodattamassa radiotaajuisia häiriöitä, kondensaattori C66 suodattamassa matalampia korkeataajuisia häiriöitä ja SMBJ33CA-piiri (D18) vaimentamassa jännitepiikkejä. Lisäksi sisääntulossa on vastuksilla R144 ja R145 toteutettu jännitteenjako, joka laskee jännitetason sopivaksi. Tämä jännite ohjaa sarjavastuksen R183 kautta komparaattorin LMV339MT sisääntuloa.

Komparaattorin tehtävänä on muuntaa jännitesignaali digitaaliseen muotoon. Kuva 3.3 on ruudunkaappaus oskilloskoopin näytöltä. Vihreällä merkitty kanava 2 mittaa funktiogeneraattorin tuottamaa kilohertsin taajuista sinimuotoista sisääntulosignaalia. Keltaisella merkitty kanava 1 mittaa komparaattorin ulostulosignaalia. Kuvasta nähdään, että sisääntulon noustessa noin 3,3 voltin suuruiseksi komparaattorin ulostulon jännite nousee noin kolmeen volttiin. Kun sisääntulon jännite laskee tämän jälkeen noin 0,9 volttiin, laskee komparaattorin ulostulojännite takaisin nollan voltin tuntumaan. Tämä hystereesi on myös komparaattorin ominaisuus [21]. Hystereesi estää sen, että jännitteen muuttuessa epätasaisesti tämä ei näy kuitenkaan digitaalisessa signaalissa ylimääräisinä tilan vaihdoksina.



Kuva 3.3. 1000Hz signaali komparaattorin läpi



Kuva 3.4. X-Y-kuvaaja komparaatorin signaalista

Kuvassa 3.4 on mitattu oskilloskoopilla X-Y-tilassa funktiogeneraattorin tuottamaa 5000Hz sinimuotoista signaalia, jonka amplitudi on 2,5 volttia. X-akselilla on funktiogeneraattorin tuottaman signaalin jännite, ja Y-akselilla on komparaattorin ulostulojännite. Hystereesi näkyy kuvassa pystysuuntaisten viivojen erotuksena.

3.4 Keskeytysohjain

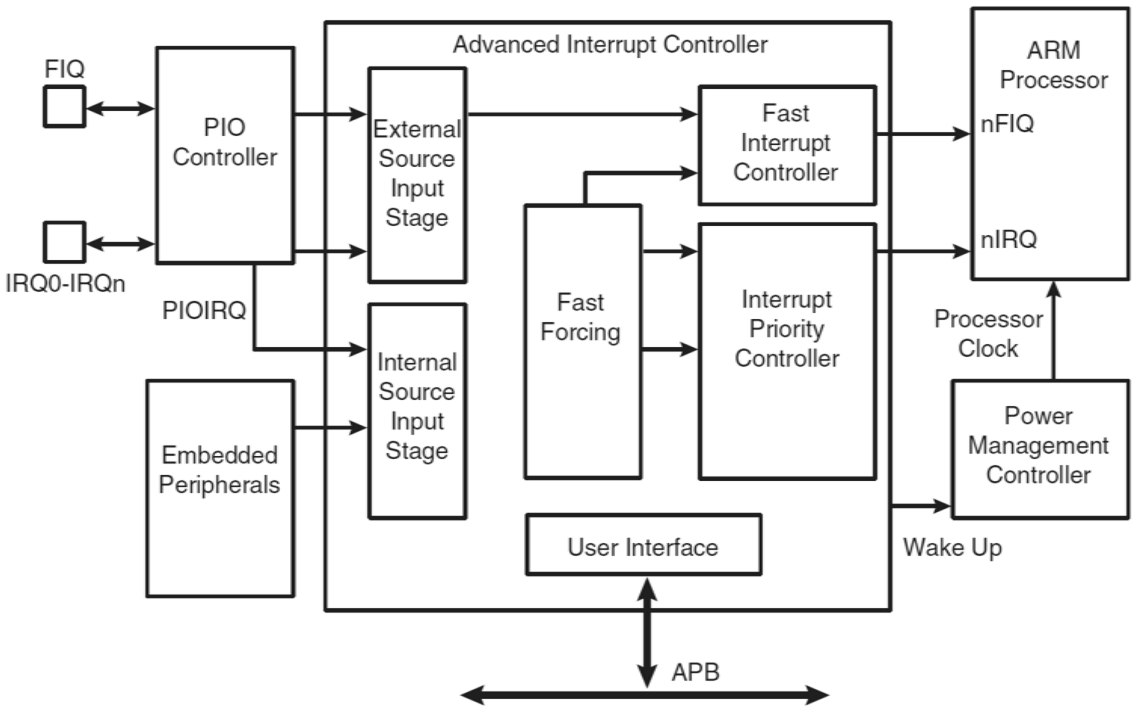
ATSAM9260:ssa on keskeytysten käsittelyä varten keskeytysohjain Advanced Interrupt Controller (AIC). Keskeytysohjain on suunniteltu vähentämään keskeytysviivettä ja suorittimen kuormaa keskeytysten käsittelyssä. Ohjaimen ulostulot on kytketty suorittimen keskeytyslinjoihin nIRQ ja nFIQ (normal IRQ ja fast IRQ). [3]

Ohjaimessa on 32 yksilöittäin maskattavaa ja vektoroitavaa keskeytyslähdettä. Näistä neljä ovat ulkoisia keskeytyslähteitä ja loput MPU:n sisäisiä. Ulkoinen lähde 0 eli Fast Interrupt Input on varattu korkean prioriteetin keskeytykselle (FIQ) Keskeytykset ovat ohjelmoitavissa reuna- tai tasoherkiksi. [3]

Kahdeksantasoinen prioriteettiohjain (Interrupt Priority Controller) mahdollistaa lähdekohtaisen prioriteetin ohjelmoinnin keskeytyslähteille 1-31 eli normaaleille keskeytyksille. Prioriteetti merkitsee käytännössä sitä, että korkeamman prioriteetin keskeytyksen käsittelijään voidaan siirtyä matalamman prioriteetin keskeytyskäsittelyn ollessa vielä kesken. Jos useampi keskeytys samalla prioriteetilla on odottamassa, käsitellään ensin se, jonka lähdenumero on pienin. [3]

Vektorointi optimoi keskeytyskäsittelijään siirtymistä ja sen suoritusta. Jokaiselle keskeytyslähteelle on 32-bittinen vektorirekisterinsä. Fast Forcing -tekniikalla tavallinen keskeytys voidaan pakottaa korkean prioriteetin keskeytykseksi. [3]

Keskeytyslinjat 2-31 voidaan kytkeä joko sisäisiin oheislaitteisiin tai ulkoisiin keskeytyslinjoihin. Ulkoiset keskeytykset voidaan kytkeä joko suoraan AIC:iin tai PIO Controlleriin (Peripheral I/O Controller). Keskeytyskäsittelyn näkökulmasta PIO Contoller nähdään käyttäjän oheislaitteina. Oheislaitteet identifioidaan keskeytyslähteen numeroinnilla. [3]

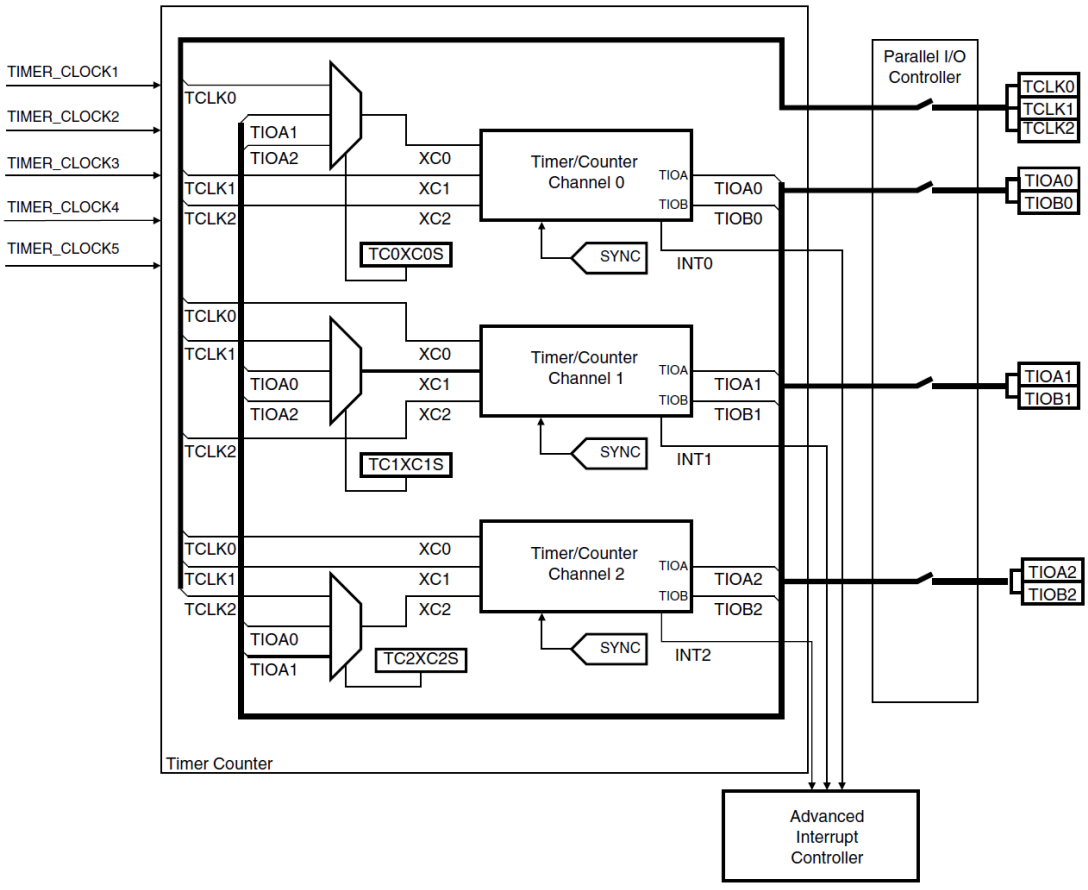


Kuva 3.5. *Keskeytysohjaimen (AIC) lohkokaavio [3]*

ATSAM9260:n keskeytyslatenssiin vaikuttavat useat asiat, kuten aika mikä ohjelmalta kuluu keskeytyksen maskaamiseen, ja suoritettavan käskyn suoritusaika. Raudan osalta keskeytyslatenssi vaihtelee liipaisutavasta riippuen kolmesta neljään ja puoleen MCK:n kellojaksoon. Ulkoisilla keskeytyksillä reunaherkän keskeytyksen latenssi on maksimissaan 4 kellojaksoa ja tasoherkän keskeytyksen 3 kellojaksoa. Sisäisillä keskeytyksillä kyseiset maksimilatenssit ovat puoli kellojaksoa suuremmat. Raudan keskeytyslatenssi on kuitenkin varsin merkityksetön, sillä suurin osa ajasta kuluu kuitenkin käyttöjärjestelmän suorittamiin toimiin. [3]

3.5 Ajastinlohko

ATSAM9260 sisältää kaksi kolmekanavaista 32-bittistä ajastinta (timer counter, TC). Yksi TC sisältää kolme identtistä kanavaa. Jokaiselle kanavalle voidaan erikseen ohjelmoida monia toimintoja, kuten taajuuden mittausta, tapahtumien laskentaa, intervallin mittausta, pulssin generointia, viiveajastusta ja pulssinleveysmodulaatiota. Jokaisella kanavalla on kolme sisääntuloa ulkoisille kellosignaaleille ja viisi sisäisille kellosignaaleille sekä kaksi käyttäjän määriteltävissä olevaa I/O-signaalia. Jokainen kanava on kytketty AIC:lle, jonka avulla voidaan generoida keskeytys suorittimelle. [3]



Kuva 3.6. *Ajastimen lohkokaavio [3]*

Kuvassa 3.6 on TC:n lohkokaavio. Oikealla on esitetty kytkennät keskeytysohjaimeen ja PIO Controlleriin sekä I/O:hin (TIO) ja ulkoisiin kellosignaaleihin (TCLK).

3.6 Varmistusrekisterit

Rekisterit koostuvat kiikuista (flip-flop). Ne tallentavat tietoa haihtuvasti (volatile), eli ne tarvitsevat sähkövirtaa arvonsa ylläpitämiseen. Rekisterit ovat hyvin nopeita sekä lukea että kirjoittaa, sillä kirjoitus- ja lukuoperaatio voidaan tehdä kellojakson aikana. Rekisterit eivät myöskään kulu käytössä, kuten flash- tai EEPROM-muisti. [32]

Suorittimen varmistusrekisterit on tarkoitettu tallentamaan tietoa, joka ei saa hävitä jos laite käy virrattomana. Tähän soveltuu myös haihtumaton muisti, kuten flash. Tällaisen muistien ongelmana on kuitenkin hidas kirjoitusnopeus, ja se että ne kuluvat käytössä. Tästä syystä niiden käyttö on ongelmallista jos arvoa pitää tallentaa jatkuvasti nopeaan tahtiin. AT91SAM9260:ssä on neljä 32-bittistä varmistusrekisteriä, joista yksi on varattu reaaliaikakellolle [3]. WRM 247:ssä on varaus maksimissaan kahdelle 20F kondensaattorille, jotka ylläpitävät muun muassa varmistusrekistereitä.

4 OHJELMISTO

4.1 Käyttöjärjestelmä

Nykyään sulautetuissa järjestelmissä on usein vähintään jonkinlainen käyttöjärjestelmä. Tarjontaa on pienistä ytimistä täysin varusteltuihin käyttöjärjestelmiin. Aikakriittisissä sulautetussa laitteessa on usein niin kutsuttu reaaliaikainen käyttöjärjestelmä (real-time operating system, RTOS) [5]. WRM247:ssä on käytetty Linux-käyttöjärjestelmää, joka ei normaalisti ole RTOS. Siihen on kuitenkin lisätty reaaliaikapäivitys (real-time patch), joka mahdollistaa käyttöjärjestelmän reaaliaikaisemman käytön.

Linux on Unix-käyttöjärjestelmään pohjautuva avoimen lähdekoodin käyttöjärjestelmä, jonka ensimmäisen version julkaisi Linus Torvalds vuonna 1991. 1990-luvun lopulla Linux alkoi saavuttaa suurempaa suosiota, ja nykyään se on yleisin käyttöjärjestelmä sulautetuissa laitteissa [6]. Tällä hetkellä uusin vakaa Linux-ytimen versio on 3.9.1 [20].

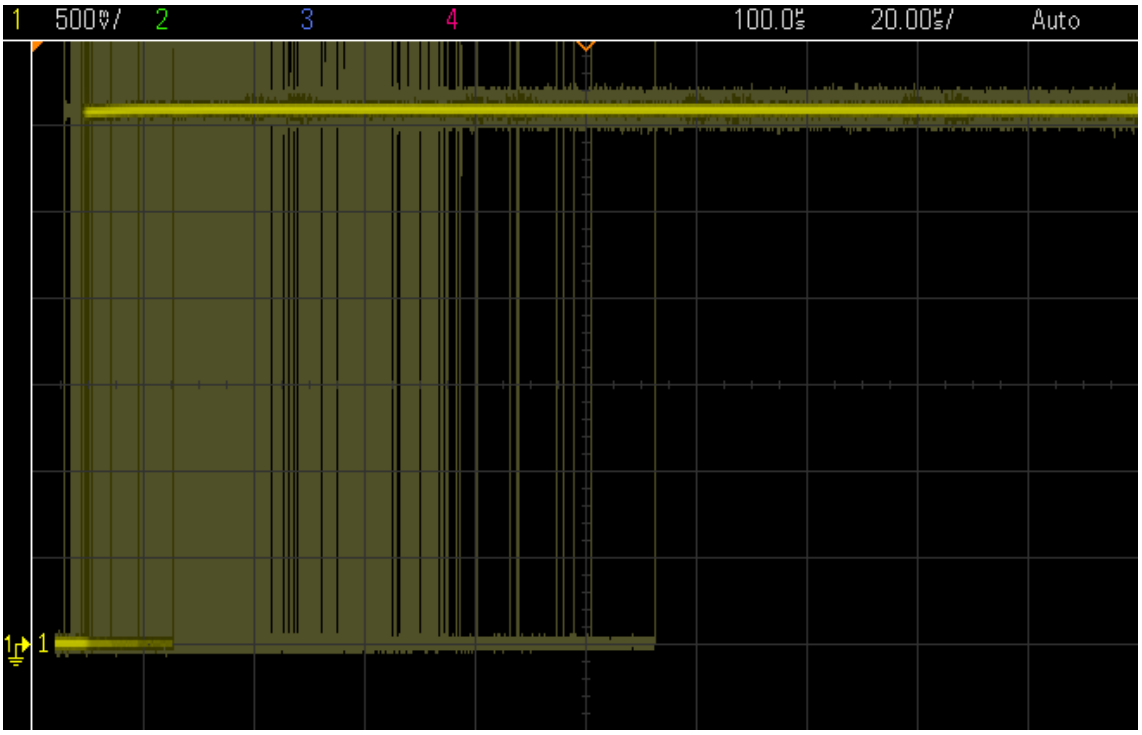
WRM:ssä käytetään Linux-ytimen versiota 2.6.29.6-rt24, missä rt24 tarkoittaa reaaliaikapäivitystä. Tämä työ käsittelee kaikilta osin tätä versiota, ellei toisin mainita. Usein Linuxilla viitataan suurempaan järjestelmään, joka sisältää käyttöjärjestelmän ytimen lisäksi ikkunointijärjestelmän ja joukon apuohjelmia. Kun tässä työssä puhutaan Linuxista, viitataan Linux-ytimeen, ellei erikseen toisin mainita.

4.2 Reaaliaikaisuus

Reaaliaikaisen järjestelmän tulee toimia sille asetettujen aikarajoitteiden puitteissa. Järjestelmän tulee siis toimia deterministisesti. Laskutoimitusten tulee olla suoritettuna tiettynä ajanhetkenä, ja järjestelmän tulee vastata tiettyihin tapahtumiin määrätyn aikaikkunan sisällä [1]. Toisin sanoen tapahtumiin pitää reagoida oikealla viiveellä, eikä pelkästään mahdollisimman nopeasti [15].

Reaaliaikavaatimukset jaetaan koviin ja pehmeisiin. Kovaa reaaliaikavaatimusta ei saa missään tapauksessa rikkoa, ja sen rikkominen voi aiheuttaa suurtakin vahinkoa. Pehmeän reaaliaikavaatimuksen rikkominen saattaa näkyä esimerkiksi käytön hitautena, mutta ei aiheuta mitään peruuttamatonta vahinkoa, joten sen rikkominen ei tavallisesti ole absoluuttisesti kiellettyä. Kovan ja pehmeän reaaliaikavaatimuksen erottaminen ei ole kuitenkaan aina ihan suoraviivaista. [15]

Reaaliaikajärjestelmiä arvioidaan yleensä kahden suureen avulla, jotka ovat latenssi ja jitter [1]. Latenssi on aikaero tapahtuman ja siihen liittyvän oikean reaktion välillä, ja Jitter kuvaa latenssin vaihtelua.



Kuva 4.1. *Latenssi ja jitter mitattuna oskilloskoopilla*

Kuvassa 4.1 on esimerkki latenssin mittauksesta oskilloskoopilla. Oskilloskooppi on asetettu piirtämään kuva siten, että se ei tyhjennä ruutua liipaisujen välissä, jolloin aikaisemmat tapahtumat jäävät näkyviin. Tapahtuma on kuvan vasemmassa reunassa. Latenssi on kuvan vasemman reunan erotus siitä hetkestä, kun signaali nousee ylös. Jitter näkyy kuvassa tämän nousuhetken vaihteluna. Tässä tapauksessa latenssi vaihtelee välillä 6-113µs, jolloin jitterin suuruus on 107µs.

4.3 RT Patch

Alunperin RTLinux-projekti julkaistiin 1996 Michael Barabanovin toimesta Victor Yodaikenin valvonnassa. Projektin tavoitteena oli tarjota deterministiset vasteajat Linux-ympäristössä. Myöhemmin projekti laajennettiin tukemaan paljon muutakin kuin alunperin tarkoitettuja sovelluksia, kuten reaaliaikaisen pörssin järjestelmiä ja muita yritys-sovelluksia. RTLinux myytiin Wind River:lle 2007-vuoden alussa. Nykyään on olemassa useampia reaaliaika-Linux-projekteja. Osa on toteutettu lisäämällä sekundäärinen ydin Linux-ytimen alle, ja osassa on paranneltu Linuxin ytimen vasteaikoja. [36]

2000-luvun ensimmäisen vuosikymmenen loppupuolella Linux-yhteisöllä oli kova pyrkimys tehdä Linux-ytimestä todellinen RTOS ilman mikroytimen apua. Tämän saavuttamiseksi on ytimeen pitänyt tehdä monia muutoksia. Jotta Linux toimisi kunnolla reaaliaikaympäristössä, keskeytyskäsittelijät eivät saa keskeyttää mitään prosessia ehdoitta, kriittisten alueiden suojaus pitää rajata estämään vain niitä prosesseja, jotka voi-

vat käyttää niitä, ja rajoittamaton prioriteettien lukkiutuminen (priority inversion) on kielletty [36]. Tällä hetkellä Linux-ytimien virallisen julkaisijan kernel.org:n uusin versio RT Patch:sta on ytimen versioon 3.6 [29]. WRM 247:ssa on tällä hetkellä RT Patch:n versio 2.6.29.6-rt24.

RT Patch:n merkittävimmät muutokset ovat seuraavanlaiset. Primitiiviset lukot – kuten spinlock – on korvattu RT-mutex:lla, mikä tekee niistä irroitettavia. Vanha Linux timer API on korvattu erillisillä rakenteilla, joilla saavutetaan korkea ajastusresoluutio. Keskeytyskäsittelijät on korvattu ytimen säikeillä (kernel thread), jolloin niitä voidaan vuorontaa ja irroittaa. [2] [27]

Alla oleva koodilistaus on WRM 247:n Linux-ytimen lähdekoodin tiedostosta kernel/irq/manage.c. Kyseinen funktio kutsutaan keskeytyksille, ellei niille ole määritelty lippua IRQF_NODELAY, jolloin keskeytys käsitellään normaalissa keskeytyskäsittelijässä, eikä sille luoda säiettä.

```
static int start_irq_thread(int irq, struct irq_desc *desc)
{
    if (desc->thread || !ok_to_create_irq_threads)
        return 0;

    init_waitqueue_head(&desc->wait_for_handler);

    desc->thread = kthread_create(do_irqd, desc, "IRQ-%d", irq);
    if (!desc->thread) {
        printk(KERN_ERR "irqd: could not create IRQ thread %d!\n", irq);
        return -ENOMEM;
    }

    /*
     * An interrupt may have come in before the thread pointer was
     * stored in desc->thread; make sure the thread gets woken up in
     * such a case:
     */
    smp_mb();
    wake_up_process(desc->thread);

    return 0;
}
```

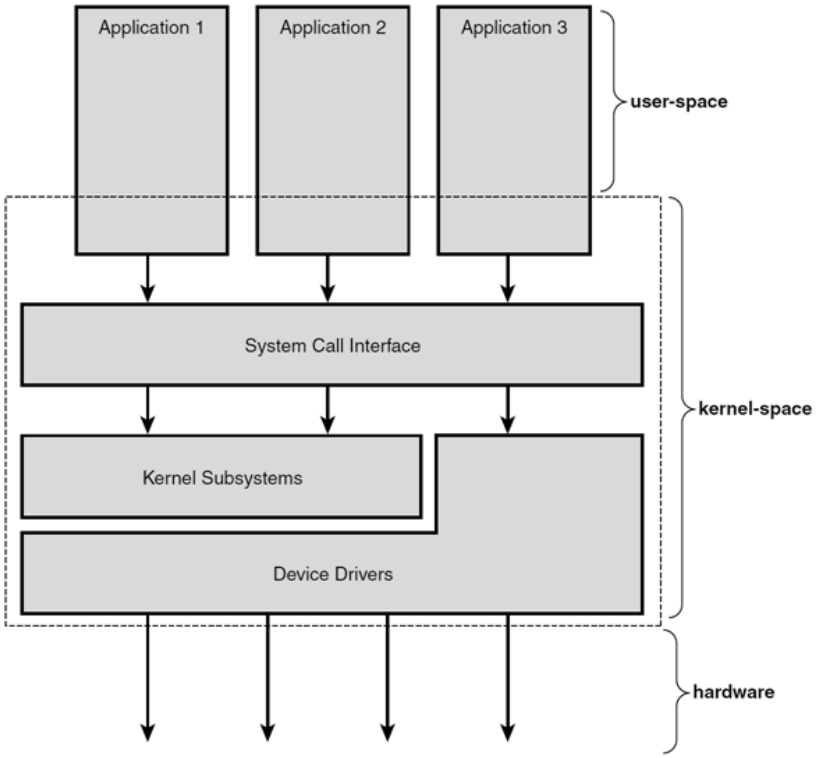
4.4 Prosessi/ydin -malli

Linux-käyttöjärjestelmässä toteutuu niin kutsuttu prosessi/ydin -malli (process/kernel model). Linux-ympäristössä suorittimella on kaksi tilaa joissa se voi toimia. Toinen on käyttäjätila (user-space, user mode) ja toinen on ydintila (kernel-space, kernel mode). Ydintilassa on vielä erityinen keskeytyskonteksti (interrupt context). Suoritinarkkitehtuureilla voi olla myös useampia tiloja; esimerkiksi 80x86-suorittimilla on neljä tilaa. Kuitenkin käyttöjärjestelmän näkökulmasta tiloja on vain edellä mainitut kaksi. [6] [22]

Kun ohjelmaa suoritetaan käyttäjätilassa, se ei voi suoraan käsitellä ytimen tietorakenteita tai sen ohjelmia. Ydintilassa suoritettavalla ohjelmalla näitä rajoituksia ei ole. Jokaisella suorittimella on erityinen käsky tilan vaihtamiseen. Tavallisesti ohjelma suoritetaan käyttäjätilassa, ja se vaihtaa ydintilaan vain tarvitessaan ytimen tarjoamia palve-

luita. Kun ydin on toteuttanut ohjelman pyynnön, se asettaa ohjelman takaisin käyttäjätilaan. [7]

Prosessi/ydin -malli olettaa, että prosessit jotka tarvitsevat ytimen palveluita, käyttävät erityistä palvelua, jota kutsutaan nimellä järjestelmäkutsu (system call). Järjestelmäkutsu asettaa ryhmän parametreja, jotka identifioivat prosessipyyntöä, minkä jälkeen suoritetaan laitteistoriippuvainen konekäsky, joka vaihtaa käyttäjätilasta ydintilaan. Järjestelmäkutsujen kautta käytetään myös laiteajureita (device driver), joilla päästään käsiksi laitteistoon. Kuva 4.2 havainnollistaa edellä mainittua mekanismia. [6] [22]



Kuva 4.2. Linuxin käyttäjätilan, ydintilan ja laitteiston yhteys *[22]*

Käyttäjätilan prosessien lisäksi Linuxissa on etuoikeutettuja prosesseja, joita kutsutaan ydinsäikeiksi (kernel thread). Ydinsäikeet suoritetaan ytimen osoiteavaruudessa ja ne eivät ole vuorovaikutuksessa käyttäjien kanssa. Ne käynnistetään yleensä järjestelmän käynnistyksen yhteydessä ja ne ovat käynnissä järjestelmän sulkemiseen asti. Ytimen rutiineja voidaan aktivoida järjestelmäkutsujen lisäksi muilla tavoilla. Suoritettava prosessi voi synnyttää poikkeussignaalin, oheislaite voi aiheuttaa keskeytyksen, tai ydinsäiettä suoritetaan. [6]

Jokaisella prosessorilla on prosessikuvaaja (process descriptor), joka sisältää tarvittavan tiedon prosessin senhetkisestä tilasta. Kun ydin pysäyttää prosessin suorituksen, se tallentaa tiettyjen suorittimen rekisterien arvot prosessikuvaajaan. Näihin rekistereihin kuuluvat ohjelmalaskuri, yleiskäyttöiset rekisterit, liukulukurekisterit, suorittimen ohjausrekisterit (processor status word) ja muistinhallintarekisterit. Kun ydin päättää palata prosessin suorittamiseen, se käyttää sopivaa prosessikuvaajan kenttää ladatakseen prosessorin rekisterit. Ohjelmalaskurin avulla ohjelman suoritusta voidaan jatkaa siitä, mihin se jäi suorituksen keskeytyessä. [6]

4.5 Virtuaalinen tiedostojärjestelmä

Yksi Linuxin valttikortteja on virtuaalisen tiedostojärjestelmän (virtual filesystem, VFS) käyttö. VFS mahdollistaa useiden eri tiedostojärjestelmien liittämisen järjestelmään siten, että niitä voi käsitellä keskenään samanlaisen rajapinnan kautta. Esimerkiksi Microsoftin NTFS-tiedostojärjestelmän tiedostoja voidaan käsitellä pääosin samoilla komennoilla kuin sulautetulla flash-muistilla olevia tiedostoja. Käyttöjärjestelmän ytimeen on sisällytetty paljon informaatiota, jonka avulla se voi käsitellä samalla tavalla monia tiedostojärjestelmiä. Ydin huolehtii siitä, että esimerkiksi luku- ja kirjoitusoperaatiot kutsuvat oikeaan tiedostojärjestelmään kuuluvia funktioita. [22]

VFS ei kata pelkästään eri tiedostojärjestelmiä, vaan se luo myös muun muassa oheislaitteille ja prosesseille rakenteet, joilla ne saadaan näkymään tiedostoina. Oheislaitteet näkyvät Linuxissa hakemistossa /dev ja prosessien tiedot hakemistossa /proc. Tätä toimintatapaa kutsutaan nimellä yleinen tiedostomalli (common file model). Perusmalli on perinteisen Unix-tiedostojärjestelmän pohjalta, ja sitä sovelletaan muihin rakenteisiin. Muut tiedostojärjestelmät, laitteiden ja prosessien tiedot muokataan noudattamaan tätä yleistä tiedostomallia. [22]

Yleinen tiedostomalli koostuu neljästä objektityypistä, jotka ovat nimeltään *superblock object*, *inode object*, *file object* ja *dentry object*. Superblock object ylläpitää informaatiota liittyen liitettyyn tiedostojärjestelmään. Inode object sisältää informaatiota tiedostosta. Jokaisella inode objectilla on yksikäsitteinen inode-numero (inode number), joka identifioi tiedoston tiedostojärjestelmän sisällä. Levypohjaisissa tiedostojärjestelmissä superblock object vastaa tavallisesti levylle tallennettua tiedostojärjestelmän kontrollilohkoa, ja inode object levylle tallennettua tiedoston kontrollilohkoa. File object sisältää informaatiota prosessin ja avoimen tiedoston välisestä vuorovaikutuksesta. Tämä tieto on olemassa ytimen muistissa vain silloin kun prosessi pitää tiedostoa auki. Dentry object sisältää tietoa hakemistomerkinnän (directory entry) – eli täydellisen tiedostonimen – ja sitä vastaavan tiedoston välillä. Tiedostojärjestelmillä on omat tapansa tallentaa tämä tieto levylle. [22]

4.6 Keskeytykset

Usein sulautetun järjestelmän toiminta pohjautuu keskeytyksiin. Järjestelmä ei välttämättä tee mitään merkityksellistä muuten kuin keskeytyksen seurauksena. Tietotekniikassa keskeytyksen (interrupt) yleinen määritelmä on tapahtuma, joka aiheuttaa muutoksen ohjelman suoritukseen. Suoritin reagoi keskeytykseen aina jollakin tavalla; se joko hylätään, tai käsitellään keskeytyskäsittelijässä. Keskeytyksen jälkeen ohjelma jatkaa suoritustaan siitä mihin se jäi ennen keskeytystä. [6]

Keskeytykset jaetaan yleensä kahteen pääryhmään: synkronisiin ja asynkronisiin keskeytyksiin. Synkroniset keskeytykset syntyvät suorittimen toimesta jonkun konekäskyn seurauksena. Ne tapahtuvat suorittimen kellon tahdissa, minkä takia niitä kutsutaan synkronisiksi. Asynkroniset keskeytykset tulevat ulkopuolisilta I/O-laitteilta kuten näp-

päimistöltä ja ne tulevat suorittimen näkökulmasta ennustamattomasti. Asynkronista keskeytystä nimitetään usein pelkäksi keskeytykseksi tai laitteistokeskeytykseksi (hardware interrupt) ja synkronista keskeytystä poikkeukseksi (exception) tai ohjelmistokeskeytykseksi (software interrupt). Terminologia vaihtelee paljon kirjallisuuslähteiden välillä. Osassa kirjallisuutta on myös poikkeuksista erotettu omaksi tyypikseen ansa (trap). Koska tämä työ käsittelee lähinnä ARM-suoritinarkkitehtuuria, tässä on käytetty ARM:n terminologian mukaisia nimityksiä keskeytys ja poikkeus. [5] [6]

Keskeytyksiä voi syntyä esimerkiksi näppäimistön näppäintä painaessa, väyläpuskurin täyttyessä tai intervalliajastimen tikityksellä. Poikkeuksia syntyy ohjelmointivirheiden seurauksena tai muuten ohjelman ajautuessa poikkeavaan tilaan. Tällainen tilanne voi olla esimerkiksi sivutusvirhe tai liukuluvun jakaminen nollalla. [6]

Keskeytys on laitteiston synnyttämä sähköinen signaali, joka on kytketty joko suoraan suorittimelle tai keskeytysohjaimelle. Keskeytysohjain on piiri joka vuorontaa useampia keskeytyslinjoja yhdeksi linjaksi suorittimelle. Linjoja voi olla myös esimerkiksi kaksi, kuten ARM-suorittimissa on normaalin keskeytyksen lisäksi korkeamman prioriteetin keskeytykselle oma linjansa. Suoritin havaitsee signaalin ja keskeyttää senhetkisen suorituksensa käsitteläkseen keskeytyksen. Suoritin voi ilmoittaa käyttöjärjestelmälle että keskeytys on tapahtunut, minkä jälkeen käyttöjärjestelmä voi siirtyä käsittelemään keskeytystä. [22]

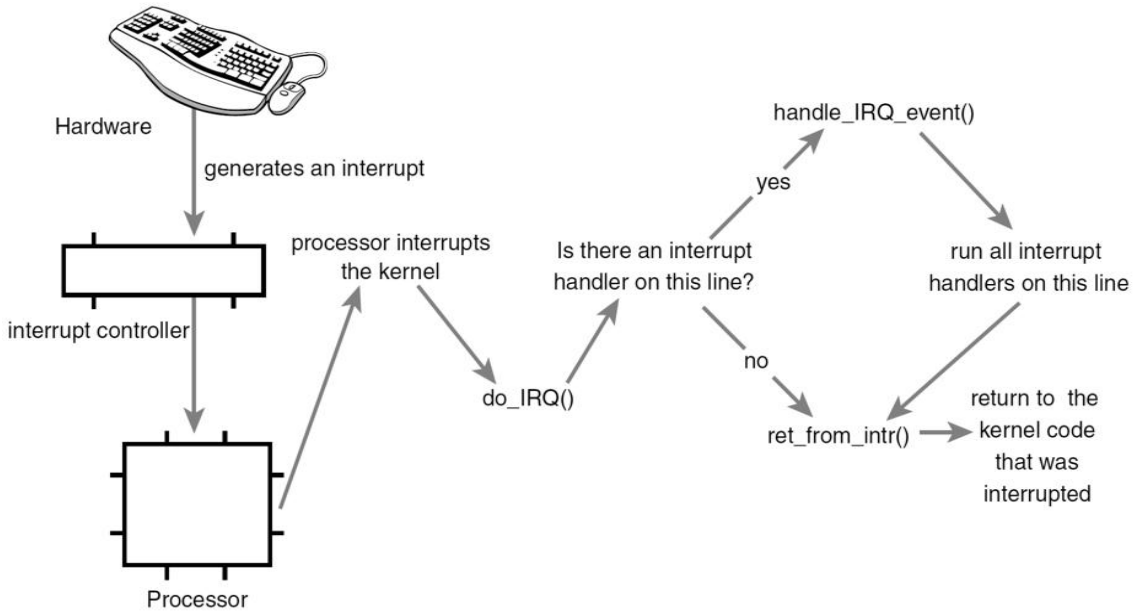
4.6.1 Keskeytyksen käsittely

Käyttöjärjestelmä erottaa keskeytykset toisistaan numeroarvolla. Tätä numeroarvoa kutsutaan tavallisesti nimellä keskeytyspyyntö (interrupt request, IRQ). Osa numeroista voi olla kiinteästi varattuja ja osa ohjelmoitavissa. Esimerkiksi ARM9-prosessorissa IRQ 0 on varattu korkeimman prioriteetin keskeytykselle ja IRQ 1 järjestelmäoheislaitteille. Arvoja voidaan myös varata dynaamisesti joissakin suoritinarkkitehtuureissa. [22]

Keskeytyskäsittelijä on aliohjelma, joka suoritetaan reaktiona keskeytykseen. Jokaiselle keskeytyksiä tuottavalle laitteelle on oma keskeytyskäsittelijänsä. Linuxissa keskeytyskäsittelijä on osana laitteen ajuria. Se on C-kielinen funktio, joka vastaa tiettyä prototyyppiä. Käyttöjärjestelmäydin antaa keskeytyskäsittelijälle tiedot keskeytyksestä standardoidulla tavalla. [22]

Käyttöjärjestelmä ajaa keskeytyksiä erityisessä kontekstissa, jota kutsutaan nimellä keskeytyskonteksti (interrupt context). Tässä kontekstissa suoritettavaa ohjelmakoodia ei voida lopettaa kesken (blokata). Tästä syystä keskeytyskäsittelijän on syytä viedä mahdollisimman vähän suoritinaikaa, jotta järjestelmän reaaliaikaisuus ei kärsisi kohtuuttomasti muiden prosessien odottaessa. Keskeytys voi tapahtua milloin tahansa, joten pitkäaikainen keskeytyskäsittely voi sattua ohjelmiston kannalta huonoon aikaan. [22]

Keskeytyskäsittelijän kokoa voidaan rajoittaa sillä, että vähemmän kiireellinen ohjelmakoodi suoritetaan keskeytyskäsittelijän ulkopuolella. Keskeytyskäsittelijä voi suorittaa vain aikakriittisimmät operaatiot ja määritellä käyttöjärjestelmälle koodin, missä suoritetaan käsittely loppuun. Keskeytyskäsittelijän pitää kuitenkin vähintään kuitata keskeytyspyyntö käsitellyksi. [22]



Kuva 4.3. *Linuxin keskeytyskäsittelyn vuokaavio [22]*

Kuvassa 4.3 on vuokaavio keskeytyksen käsittelystä Linuxissa. Laitteistotasolla laite synnyttää keskeytyksen, jonka keskeytysohjain käsittelee, ja lähettää keskeytyssignaalin suorittimelle. Suoritin keskeyttää ytimen ja siirtyy käsittelemään keskeytyspyyntöä funktiossa *do_IRQ()*. Jos kyseiselle keskeytyspyynnölle on olemassa keskeytyskäsittelijä, siirtyy suoritus funktioon *handle_IRQ_event()*, joka kutsuu keskeytyskäsittelijää. Tämän jälkeen funktio käynnistää säikeen, jos keskeytyskäsittelijässä on näin asetettu. Lopuksi palataan suorittamaan ytimen koodiin, mitä oltiin ajamassa ennen keskeytyksen käsittelyä. [22]

4.6.2 Linuxin /proc -rajapinta

Kun Linuxissa suorittimelle tulee keskeytys, kasvatetaan sisäistä laskuria, jonka avulla voidaan tarkistaa, että laite toimii odotetusti, eli keskeytyksiä ei katoa, eikä niitä synny ylimääräisiä [9]. Raportoidut keskeytykset näkyvät virtuaalisen tiedoston */proc/interrupts* sisältönä. Alla on esmerkki ARM-suorittimella varustellun Raspberry Pi -tietokoneen [26] */proc/interrupts* -tiedoston sisällöstä.

```
           CPU0
  3:   37451315   ARMCTRL  BCM2708 Timer Tick
 32:  780967491   ARMCTRL  dwc_otg, dwc_otg_pcd, dwc_otg_hcd:usb1
 52:          0   ARMCTRL  BCM2708 GPIO catchall handler
 65:    5366327   ARMCTRL  ARM Mailbox IRQ
 66:          1   ARMCTRL  VCHIQ doorbell
 75:          1   ARMCTRL
 77:     275456   ARMCTRL  bcm2708_sdhci (dma)
 83:         20   ARMCTRL  uart-pl011
 84:    3484519   ARMCTRL  mmc0
FIQ:              usb_fiq
Err:          0
```

Ensimmäinen sarake on IRQ-numero. Numeroista näkyvät vain ne, jotka ovat käytössä tiedoston tarkasteluhetkellä. Alimpana on numeron tilalla *FIQ*, mikä on korkean prioriteetin keskeytys (fast interrupt) ARM-arkkitehtuurissa. [9]

Seuraavissa sarakkeissa on keskeytysten määrä, jotka kyseinen suoritin on ottanut vastaan. Tässä tapauksessa suorittimia on vain yksi, joten sarakkeitakin on vain yksi. Seuraava sarake antaa tietoa keskeytysohjaimelle, ja viimeisessä sarakkeessa on laitteiden nimet, joille on rekisteröity keskeytyskäsittelijä *request_irq()*-funktion yhteydessä. IRQ:n 32 kohdalla on useampi nimi, koska se on jaettu keskeytys. [9]

Toinen keskeytyksistä tietoa antava tiedosto on */proc/stat*, jossa näkyy myös niiden keskeytysten määrä, joiden keskeytyskäsittelijä ei ole käytössä tiedoston tarkasteluhetkellä [9]. Tämä voi olla havainnollisempi silloin, jos laiteajurin keskeytyskäsittelijät vapautetaan välillä.

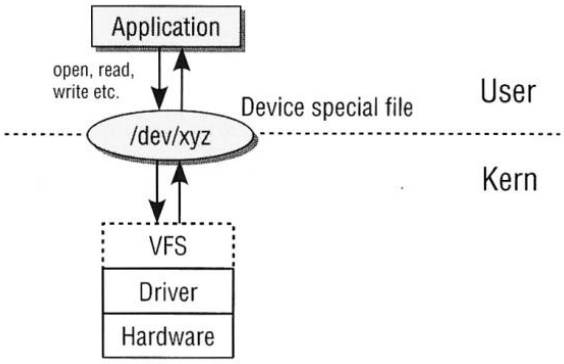
4.7 Laiteajurit

Laiteajuri (device driver) on joukko ytimen rutiineja, joka saa laitteen vastaamaan ohjelmointirajapintaan. Linuxissa laiteajureita ohjataan VFS:n avulla. Jokaisella laiteajurilla on oma laitetiedostonsa (device file) hakemistossa */dev*. Tavallisesti jokaisella laitteella on oma laiteajurinsa, sillä yleensä laitteiden I/O-kontrollerit eroavat toisistaan, ja niillä on erilaiset komennot ja tilainformaatio. Ajurikoodi sisältää toteutukset tarpeellisista VFS-funktioista, kuten *open*, *read*, *lseek* ja *ioctl*. [6]

Linuxissa oheislaitteet on jaettu kolmeen luokkaan, joita ovat merkkilaitteet (character devices), lohkolaitteet (block devices) ja verkkolaitteet (network devices). Merkkilaitteiden dataa käsitellään peräkkäisinä tavuina virtana. Merkkilaitteita ovat esimerkiksi näppäimistö ja sarjaväylä. Sovellukset toimivat merkkilaitteen kanssa suoraan laitesolmujen (device node) avulla. Lohkolaitteiden dataa käsitellään lohkoittain. Ne tuke-

vat datan etsimistä ja satunnaista hakua (random access). Lohkolaitteita ovat esimerkiksi massamuistit kuten kiintolevyt, optiset levyt ja flash-muistit. Verkkolaitteet tarjoavat pääsyn verkkoon, kuten Internetiin. Ne käyttävät fyysistä verkkosovitinta ja erityistä protokollaa kuten IP (Internet Protocol). Verkkolaitteet poikkeavat muista luokista siten, että niillä ei ole omaa laitetiedostoa. [6] [22]

Kuva 4.4 havainnollistaa laitteiston käyttöä eri tasojen läpi. Käyttäjätilan sovellus käyttää laitetiedostoa järjestelmäkutsuilla hakemistosta */dev*. Laitetiedosto toimii rajapintana laiteajuriin VFS:n avulla, ja laiteajuri kommunikoi laitteiston kanssa. Käyttäjätilassa (kuvassa User) siis tehdään vain tiedosto-operaatioita ja muu toiminnallisuus on ydintilassa (kuvassa Kern). [23]



Kuva 4.4. *Tasokuva oheislaitteen käytöstä [23]*

Jos oheislaite käyttää keskeytyspalvelua, sijaitsee keskeytyskäsittelijä laiteajurin koodissa. Keskeytyskäsittelijä tulee liittää IRQ:n sen rekisteröinnin yhteydessä. IRQ rekisteröidään laiteajurin lataamisen yhteydessä, ja vapautetaan sen vapauttamisen yhteydessä. Nämä toiminnot suoritetaan seuraavilla funktioilla:

```
request_irq(unsigned int irq,
    irq_handler_t handler,
    unsigned long flags,
    const char *name,
    void *dev)

free_irq(unsigned int irq,
    void *dev)
```

Request_irq ottaa parametrina keskeytysrivin numeron *(irq)*, funktio-osoittimen keskeytyskäsittelijään *(handler)*, mahdolliset liput *(flags)* ja nimen *(name)*, jolla IRQ esiintyy esimerkiksi tiedostossa /proc/interrupts. [22] Lipuista tämän työn kannalta merkittävin on RT Patch:n mukanaan tuoma IRQF_NODELAY. Normaalisti RT-päivitetty Linux tekee keskeytyskäsittelijästä ydinsäikeen, mutta jos IRQF_NODELAY -lippu on ylhäällä IRQ:n rekisteröinnissä, tehdään siitä normaali keskeytyskäsittelijä. [1]

5 TOTEUTUSNÄKÖKULMIA

5.1 Keskeytysten käyttö

Keskeytysten käsittely on varsin keskeinen osa pulssilaskurin laiteajuria suunnitellessa. Järjestelmän on pystyttävä tiettyyn reaaliaikaisuuteen jatkuvista keskeytyksistä huolimatta. Kappaleessa 4.2 määriteltiin reaaliaikaisuutta mittaavat suureet latenssi ja jitter. Keskeytyslatenssin määritelmä on aikaero tapahtuman signaloinnista siihen hetkeen, kun koodia aletaan suorittaa [2].

Normaalisti RT Patch Linuxissa keskeytykset ovat ytimen säikeitä. Kuitenkin nopeiden keskeytysten yhteydessä voi olla tarvetta käyttää keskeytyskäsittelijän määrittelyssä IRQF_NODELAY -lippua, jolloin keskeytyskäsittelijästä ei tehdä säiettä, vaan se käsitellään tavallisessa keskeytyskäsittelijässä [27]. Lipun käyttämisen tarve selvitettiin testaamalla järjestelmää suurella kuormalla ja korkealla mitattavan signaalin taajuudella.

5.2 Kierrosluvun laskenta

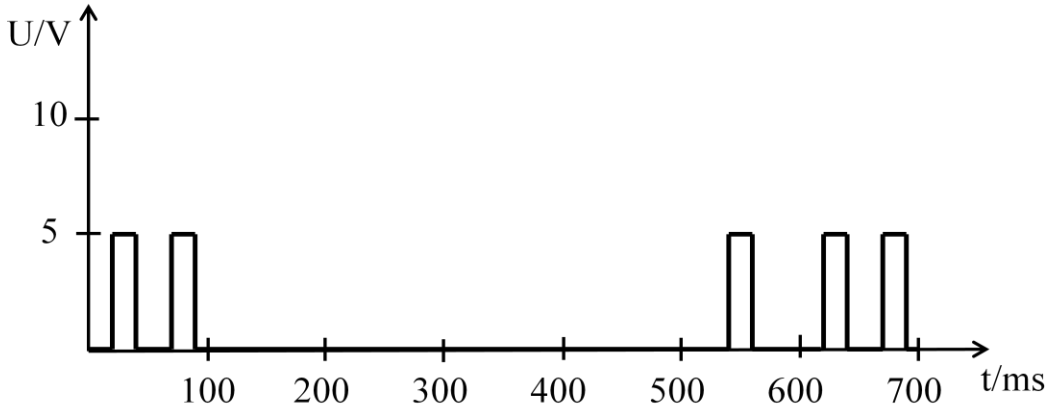
Moventas Gears:n sovelluksen vaatimuksena on, että käyttäjätilan ohjelma lukee kierrosnopeuden arvon 100ms väliajoin laiteajurilta. Ajurin on palautettava lukuoperaation tuloksena viimeisin mittaamansa kierrosnopeuden arvo. Näistä lasketaan 20 viimeisimmän näytteen keskiarvo, joka välitetään PC-sovellukselle 2 sekunnin välein. Kierrosluvun laskenta tehdään laiteajurissa ja keskiarvon laskenta käyttäjätilan sovelluksessa.

5.3 Keskiarvon laskenta

PC-sovellusta varten keskiarvo lasketaan viimeiseltä kahdelta sekunnilta. Näytteistystaajuus on 10Hz, jolloin kahden sekunnin jakso sisältää 20 näytettä. Keskiarvon laskemisessa pitää ottaa huomioon muutamia asioita. Näytteet saattavat olla tyhjiä, jos yhtään pulssia ei ole tullut edellisen näytteen jälkeen. Tällöin keskiarvo voitaisiin laskea siten, että tyhjät näytteet jätettäisiin kokonaan pois keskiarvon laskusta. Tällöin kuitenkin keskiarvoon tulee merkittävä virhe tietyissä tapauksissa. Myöskään näiden korvaaminen nollalla ei anna oikeaa tulosta.

Esimerkiksi kuvan 5-1 signaalista voitaisiin saada näytteet, jotka on esitetty taulukossa 5.1. Jos tästä laskettaisiin keskiarvo jättämällä tyhjät arvot pois laskennasta, saataisiin tulokseksi (1200 + 110 + 1000) / 3 = 770. Tällöin kuitenkin jätetään huomioimatta se, että todellisuudessa kierrosnopeus on ollut 110 myös 100ms ja 500ms välillä, eikä vain 500ms ja 600ms välillä. Keskiarvo pitää siis painottaa pulssin pituuden approksi-

maatiolla, jotta keskiarvo olisi enemmän oikein aika-akselin suhteen. Esimerkkitapauksessa arvon 110 painoarvo olisi 5 ja muiden arvojen 1. Tällöin keskiarvosta saadaan (1200*1 + 110*5 + 1000*1) / 7 = 393. Ero on tällöin varsin merkittävä painottamattomaan keskiarvoon nähden.

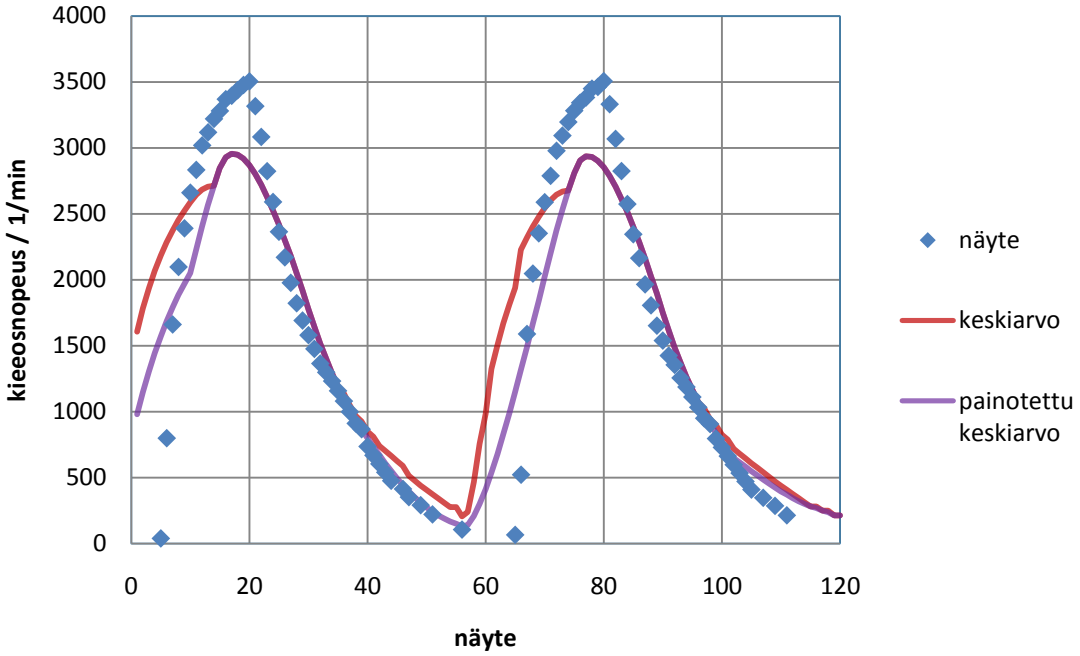


Kuva 5-1. *Esimerkki kierrosnopeussignaalista*

Taulukko 5.1. *Esimerkkisignaalista otetut näytteet*

Näyte	1	2	3	4	5	6	7
Arvo (rpm)	1200	-	-	-	-	110	1000

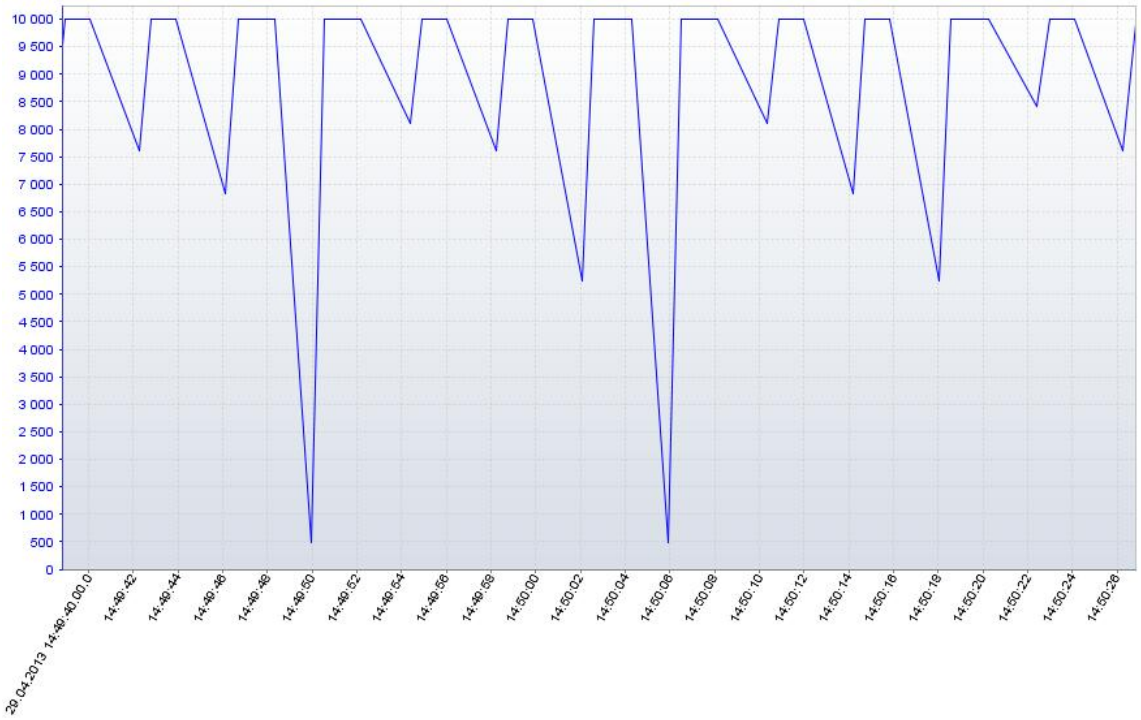
Kuvassa 5.2 on testikokoonpanosta mitattua dataa. Mitatut näytteet on kuvattu vinoneliöillä. Sinisellä viivalla on kuvattu näytteistä laskettu suora keskiarvo ja punaisella on kuvattu edellä mainitulla tavalla laskettu painotettu keskiarvo. Keskiarvot on laskettu 20 näytteestä siten, että siihen on otettu mukaan 10 edellistä ja 10 seuraavaa näytettä. Keskiarvossa näkyy selvä notkahdus siinä vaiheessa kun käyrä on saavuttamassa huippuaan. Tämä johtuu siitä, että nopeaa nousua edeltävät tyhjät näytteet vääristävät suoraa keskiarvoa. Painotetun keskiarvon käyrämuoto mukailee tarkemmin näytemateriaalia, eikä siinä näy suuria ylimääräisiä notkahduksia.



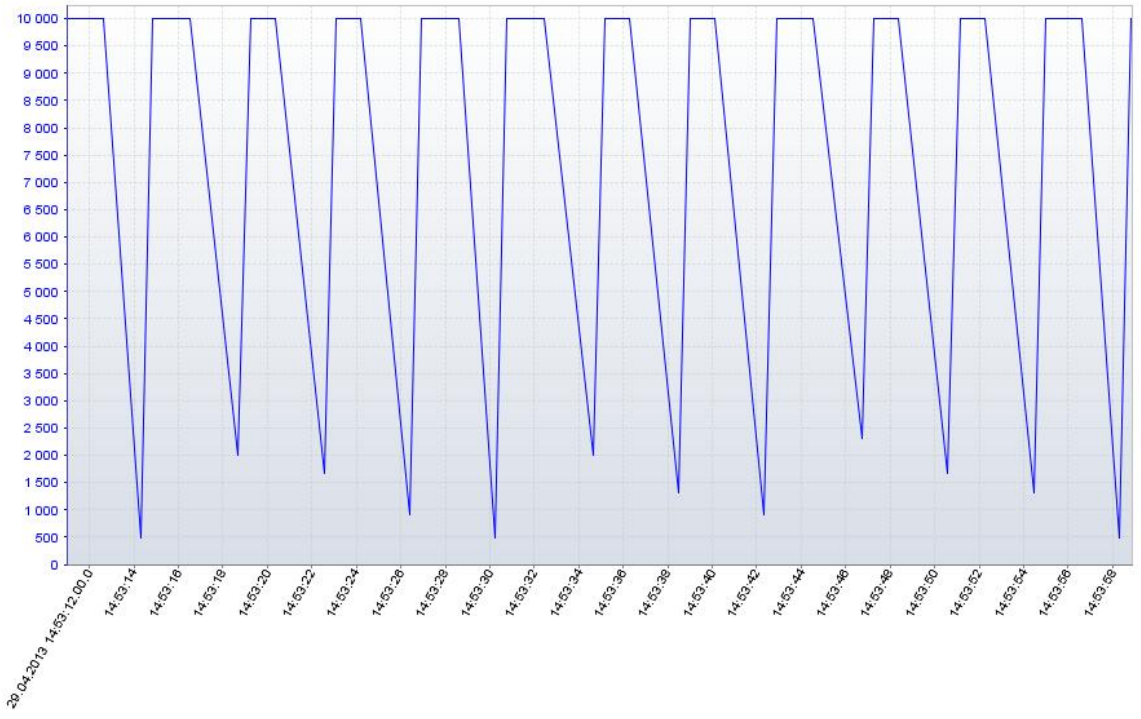
***Kuva 5.2**. Esimerkki näytteiden keskiarvostamisesta*

Kuvissa 5.3 ja 5.4 on ruudunkaappaus WRM:n käyttöliittymästä. Ensimmäisessä on laskettu suoraa keskiarvoa ja jälkimmäisessä painotettua keskiarvoa samasta signaalista. Signaali on 10kHz kanttiaaltoa, jota kytketään kahden sekunnin välein päälle ja pois päältä. Toisin sanoen signaalissa on sekoitettu keskenään 10kHz ja 4Hz kanttiaallot. Puskuri on luettu 550ms välein, ja puskurin sisällöstä on laskettu keskiarvo. 550ms on valittu siksi, että tällöin puskuria luetaan 4Hz signaalin jakson eri vaiheissa.

Kuvassa 5.3 taajuusarvoissa on selvästi enemmän vaihteluita ajanhetkinä, joissa signaalia ei ole kytketty. Kuvassa 5.4 vaihtelu on vähäisempää, ja arvot ovat keskimäärin pienempiä.



Kuva 5.3. *Pätkitetyn 10kHz signaalin keskiarvoja*



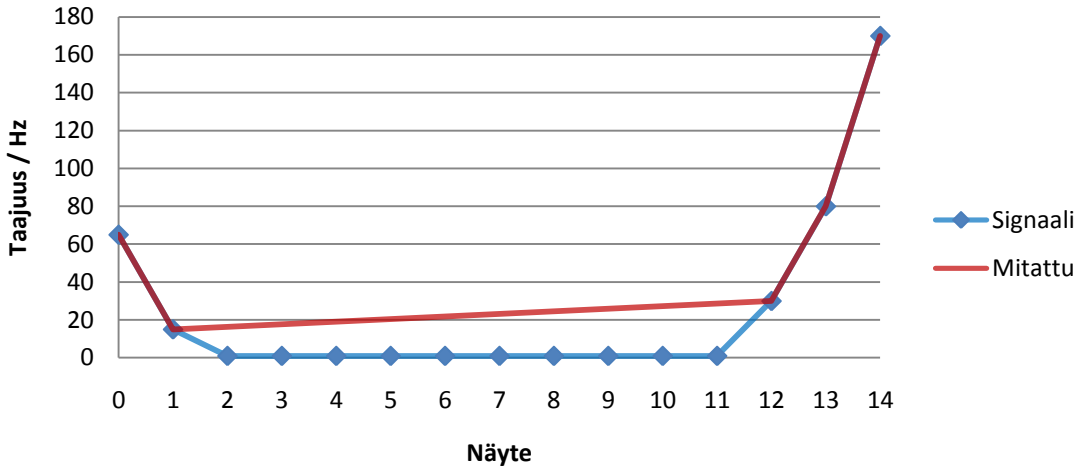
Kuva 5.4. *Pätkitetyn 10kHz signaalin painotettuja keskiarvoja*

Testitapauksen signaali on keinotekoinen, mutta tuo hyvin ilmi keskiarvon painottamisen vaikutuksen. Käytännössä kierrosnopeutta mitatessa pyörivällä kappaleella on aina jokin hitausmomentti, joten painotetun keskiarvon merkitys ei ole niin suuri. Myös keskiarvoalgoritmin valinta on sovelluskohtainen riippuen siitä, mitä todellisuudessa halutaan mitata.

5.4 Menetetyn datan minimointi

Laiteajuri palauttaa normaalisti viimeisimmän mittaamansa kierrosnopeuden arvon. Tämä menetelmä ei ole kaikissa tapauksissa optimaalinen tapa toimia. Tilanteessa, jossa kierrosnopeus nousee äkillisesti, saatetaan huonossa tapauksessa menettää dataa pitkältäkin aikaväliltä.

Tarkastellaan kuvan 5.5 esimerkkiä, jossa pulssimuotoista signaalia on näytteistetty 100ms mittausvälillä. Näytteiden 1 ja 2 välissä taajuus putoaa 1 hertsiin, jossa se pysyy näytteeseen 11 asti. Tällä 900 millisekunnin aikavälillä ei synny yhtään pulssia. Näytteiden 11 ja 12 välillä taajuus nousee nopeasti, jolloin mittausvälillä tapahtuu useita pulsseja. Laiteajuri palauttaa taajuustiedon vain kahden viimeisimmän pulssin aikaerosta, jolloin näytteen 12 arvoksi tulee 50Hz. Tällöin minkään näytteen arvoksi ei tule 1Hz, vaikka taajuus on ollut 1Hz yli 900 millisekunnin ajan. Kuvan 5.5 punainen viiva havainnollistaa tätä. Tällä tavoin mitattuna näytteiden 0-15 keskiarvoksi saadaan 72Hz, vaikka todellinen keskiarvo on noin 25Hz.



Kuva 5.5. *Esimerkki kierrosnopeuden näytteistyksestä*

Virhe saadaan korjattua siten, että tällaisessa tapauksessa laiteajuri palauttaakin viimeisimmän arvon sijaan ensimmäisen arvon edellisen mittauksen jälkeen. Eli jos edellinen näyte on ollut tyhjä ja sen jälkeen saadaan useampi pulssi, ajuri palauttaa ensimmäisen arvon. Tällöin menetettäisiin kuitenkin vielä esimerkkitapauksen näytteen 12 arvo 50Hz. Tämä on ratkaistu siten, että sovellus havaitsee tällaisen tilanteen, ja se kysyy välittömästi viimeisimmän arvon. Ensimmäinen arvo sijoitetaan näytepuskurissa edelliseen kohtaan, eli tässä tapauksessa näytteen 11 kohdalle. Tästä saadaan myös kappaleen 5.3 menetelmällä generoitua arvo näytteille 2-10.

5.5 Mittauksen tarkkuus

Laiteajuri ei merkitse mittauksiin aikaleimoja, vaan antaa ainoastaan viimeisimmän arvon lukuun ottamatta kappaleessa 5.4 määriteltyä poikkeustapausta. Kun ajurilta luetaan arvo, on lukuhetken ja viimeisimmän pulssin välillä aina aikaeroa, jota ei voida kuitenkaan kompensoida aikaleimojen puuttuessa. Tämä aiheuttaa mittausvirhettä aikatasossa. Tämän virheen suuruus on

$$\Delta t < \frac{1}{f} < T, \tag{5-3}$$

missä Δt on virheen suuruus ja T on näytteistyksen jaksonaika. Tarkkuus on siis parempi suurilla kierrosnopeuksilla ja suurella näytteenottotaajuudella. Moventas Gears:n tilaamassa sovelluksessa tapauksessa tämä epätarkkuus on hyväksyttävää, sillä käyttäjälle kierrosnopeudesta näytetään vain keskiarvo kahden sekunnin ajalta. Kappaleessa 6.1 esiteltävältä MPPC-laiteajurilta on mahdollisuus lukea tämänhetkinen laskurin arvo, jolloin edellä mainittu epätarkkuus on mahdollista kompensoida jossain määrin.

Mitä enemmän pulsseja saadaan mittauksen aikana, sitä tarkempi mittaus on. Ei voida varmasti tietää, mitä kahden pulssin välillä on tapahtunut. Jos esimerkiksi saadaan kaksi pulssia 5 sekunnin aikavälillä, ei voida tietää, onko akseli pyörinyt koko ajan tasaisesti 0,2Hz taajuudella, vai onko se pysähtynyt jossain välissä. Tällöin pyörimiskäytöstä on tehtävä joitakin oletuksia. Jos mitataan kierrosnopeutta esimerkiksi tuuliturbiinin vaihteesta, voidaan olettaa, että kiihtyvyydet pysyvät tietyissä rajoissa, sillä pyörivillä kappaleilla on tässä runsaasti hitausmomenttia.

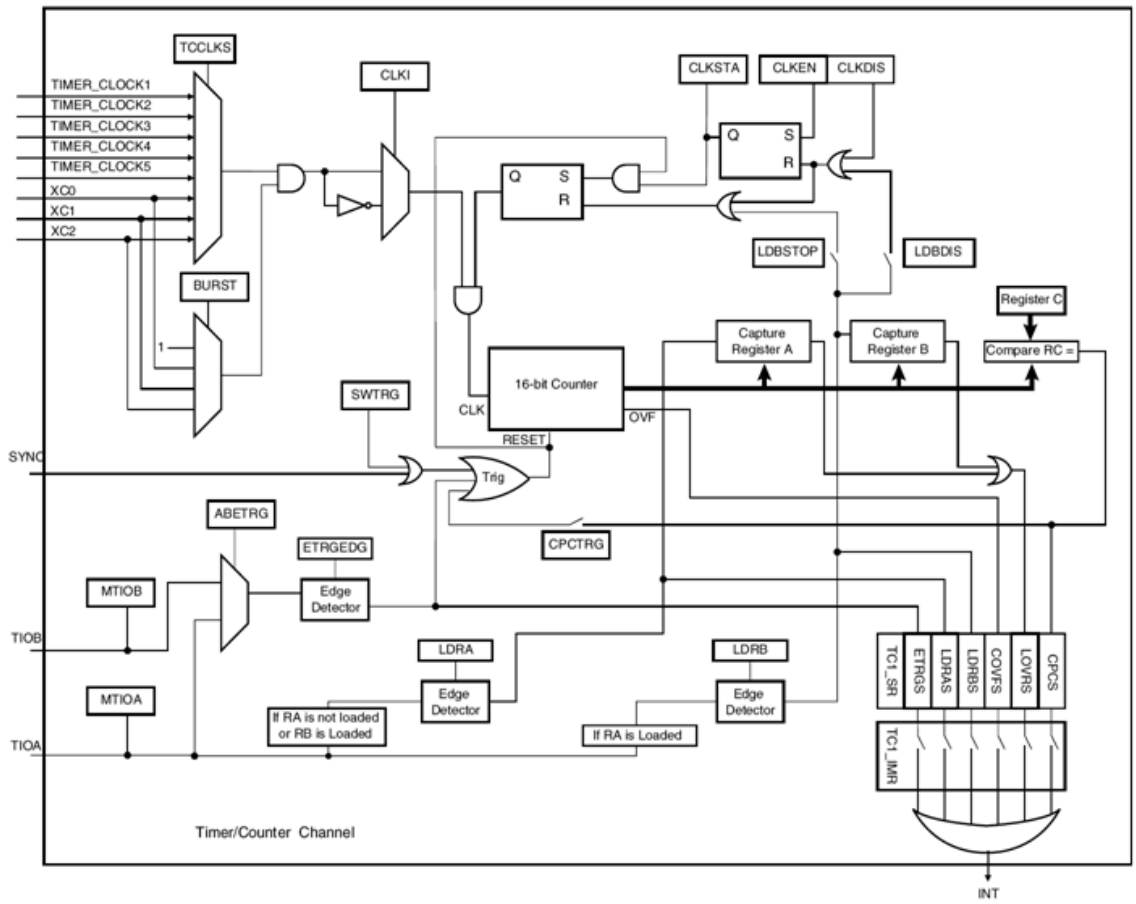
Taajuuslukeman tarkkuutta on vaikea arvioida ennalta, sillä siihen vaikuttavat monet tekijät niin laitteistossa kuin ohjelmistossakin. Latenssit eivät sinänsä vaikuta taajuuden tarkkuuteen, mutta latenssin jitter vaikuttaa, jos sitä ei saada kompensoitua.

6 OHJELMISTOTOTEUTUS

6.1 Laiteajuri

Pulssilaskurin laiteajurista tehtiin kaksi versiota. Ensimmäinen tehtiin Moventas Gears:n tarpeisiin kierrosnopeuslaskuriksi. Kun tähän vaadittu toiminnallisuus oli saavutettu, laajennettiin ajurin toiminnallisuutta monikäyttöisemmäksi. Ajurin ensimmäinen versio nimettiin lyhyesti RPM ja laajennetun version nimeksi tuli MPPC (Multi-Purpose Pulse Counter).

Laiteajuri käyttää MPU:n kahta 16-bittistä TC-lohkoa, joita kutsutaan tässä nimillä TC1 ja TC2. Molemmat TC-lohkot on asetettu *capture*-tilaan. Kuvassa 6-1 on capture-osan lohkokaavio. TC1 mittaa aikaa ja on jatkuvasti käynnissä. TC2:a käytetään pulssin reunan tunnistamiseen ja keskeytyslatenssin kompensointiin. Tarkasteltava signaali on kytketty TC2:n sisääntuloon TIOB, joka on WRM 247:n digitaalinen sisääntulo 1.



***Kuva 6-1**. Ajastimen capture-osan lohkokaavio [3]*

TC1 käyttää MPU:n sisäistä kelloa *TIMER_CLOCK5*, jonka taajuus on MCK/128 = 781kHz [3]. Kellon esijakajaksi on valittu suurin mahdollinen, jotta se ylivuotaisi mahdollisimman harvoin. Ylivuoto pitää kuitenkin huomioida omalla keskeytyksellä, jossa kasvatetaan keskeytyslaskurin arvoa. TC2:n kellona on *TIMER_CLOCK4*, joka taajuus on MCK/32 = 3125kHz [3]. Tämä ylivuotaa noin 21 millisekunnissa, mikä riittää latenssin mittaamiseen ylivuodoitta. TC2 on asetettu nollaamaan laskurinsa aina, kun se havaitsee pulssinreunan.

Kun TC2 havaitsee nousevan pulssinreunan sisääntulossa, se aiheuttaa keskeytyksen. Keskeytyskäsittelijässä otetaan ensimmäisenä talteen molempien laskurien arvot. Tämän jälkeen tarkastetaan, onko TC1 ylivuotanut keskeytykseen siirtymisen aikana ja sen mukaan tarvittaessa kasvatetaan ylivuotolaskurin arvoa. TC1:n, TC2:n ja ylivuotolaskurin avulla lasketaan pulssinreunan tapahtumahetki t_n seuraavasti:

$$t_n = \left((C_{TC1} + C_{ylivuoto} * 2^{16}) * 4 - C_{TC2}\right), \quad (6\text{-}2)$$

missä C tarkoittaa laskurin arvoa. Tästä tuloksesta vähennetään kappaleessa 2.1 esitellyn lausekkeen 1 mukaisesti edellisen pulssinreunan aika t_{n-1}. Laskutoimituksen tuloksena saadaan jaksonaika *T* laskurin TC2 askeleina. Tämä arvo muutetaan lukuoperaation yhteydessä haluttuun muotoon.

6.1.1 Versio RPM

Laiteajurin RPM-versio palauttaa lukuoperaation tuloksena yhden taajuusarvon, joka on yleensä viimeisin laskettu arvo. Kappaleen 5.4 tapausta varten lukuoperaatio ei kuitenkaan palauta viimeisintä arvoa ihan jokaisessa tapauksessa. Jos kyseistä lukuoperaatiota on edeltänyt vähintäänkin yksi tyhjä lukuoperaatio, palautetaan edellistä lukuoperaatiota seuranneen ensimmäinen pulssin taajuusarvo viimeisen pulssin arvon sijaan. Tällöin pulssia, jonka jaksonaika on pidempi kuin käyttäjätilasovelluksen kyselyväli, ei menetetä missään tapauksessa.

RPM-laiteajurin funktiolistaus on liitteessä A. VFS-funktioista on toteutettu *read*, *open* ja *release*. *Open*-funktiossa laskurit asetetaan käyntiin, ja release-funktiossa ne pysäytetään. Tällöin ei aiheuteta turhia keskeytyksiä silloin, kun ajuria ei käytetä. *Read*-funktio laskee viimeisimmästä jaksonajasta kierrosnopeuden arvon, ja välittää sen käyttäjätilan puolelle. Jos edellisen lukuoperaation jälkeen ei ole tullut uusia pulsseja, funktio palauttaa yhä viimeisimmän mittaamansa arvon, mutta se ilmoittaa lipulla, että uutta arvoa ei ole tullut. Tällöin käyttäjätilan sovellus voi tulkita tällaisen arvon tyhjäksi.

6.1.2 Versio MPPC

MPPC-versiossa on kolme toimintatilaa: pulssilaskuri, taajuus ja pulssin leveys. Tila määrittelee sen, missä muodossa lukuoperaatio palauttaa arvon. Pulssilaskuritila palauttaa mitattujen tapahtumien määrän. Taajuustila palauttaa tapahtumien taajuuden sille asetetulla kertojalla kerrottuna. Pulssinleveystila palauttaa kahden tapahtuman välisen

ajan mikrosekunteina. Tapahtuma on asetettavissa pulssin nousevaksi, laskevaksi tai molemmiksi reunoiksi.

Taajuus- ja pulssinleveystilassa on käytössä rengaspuskuri (ring buffer) [28], johon tallennetaan jokainen mitattu arvo. Puskurista poistetaan arvoja sitä mukaan kun niitä luetaan. Puskuri varataan dynaamisesti, ja sen maksimikoko on muutettavissa ajonaikaisesti maksimissaan 64 kilonäytteeseen. Näyte sisältää 32-bittisen arvon ja 32-bittisen muuttujan lipuille. Lipuilla ilmaistaan muun muassa se, että puskuri on täynnä tai tyhjä, tai sieltä luettiin viimeinen arvo. Puskurin koko voidaan kysyä *ioctl*-komennolla, jolloin se voidaan lukea tyhjäksi yhdellä *read*-operaatiolla.

Read-operaatio pulssilaskuritilassa ei nollaa pulssilaskurin arvoa. Laskurin arvo voidaan asettaa *write*-operaatiolla, jolla lukema voidaan myös nollata. Tällöin laskuri voidaan nollata vasta silloin, kun arvon säilyminen on varmistettu. Tämä järjestely takaa sen, että laskurin arvo ei häviä esimerkiksi siinä tapauksessa, että laitteen virta katkeaa välittömästi lukuoperaation jälkeen, ja lukemaa ei ole ehditty vielä tallentaa haihtumattomaan muistiin tai lähettää palvelimelle.

MPPC käyttää MPU:n varmistusrekistereitä tallentamaan pulssilukemaa ja asetuksiaan. Kummallekin on varattu yksi 32-bittinen rekisteri. Asetusten rekisteriin on lisätty tarkistussumma, jonka avulla voidaan havaita, jos rekisterin arvo on korruptoitunut. Kun varmistusrekisterit ovat käytössä, ovat pulssilukema ja asetukset tallessa virtojen katketessa. Pulssilaskenta myös käynnistyy uudestaan heti, kun ajuri alustetaan laitteen käynnistyksen yhteydessä. Tällä pyritään minimoimaan hukattujen pulssien määrä sinä aikana, kun sovellusohjelma ei ole käynnissä.

MPPC-ajurissa VFS-funktioista on toteutettu *read*, *write*, *ioctl*, *open* ja *release*. *Ioctl*-funktion komennot on esitetty taulukossa 6.1.

Taulukko 6.1. *MPPC-ajurin ioctl-komennot*

ioctl-komento	Toiminto
`MPPC_SET_MODE`	Asettaa ajurin toimintatilan.
`MPPC_SET_MULTIPLIER`	Asettaa taajuudelle kertojan.
`MPPC_SET_BUFFER_SIZE`	Asettaa näytepuskurin koon, jos koko on alle 1024.
`MPPC_SET_BUFFER_SIZE_K`	Asettaa näytepuskurin koon kilonäytteinä (1024 näytettä), jos koko on 1024 tai enemmän.
`MPPC_SET_USE_GPBR`	Jos arvo on 1, ottaa käyttöön varmistusrekisterit (GPBR).
`MPPC_SET_EDGE`	Asettaa signaalin reunan, joka liipaisee mittauksen.
`MPPC_GET_PULSE_COUNT`	Palauttaa pulssilukeman.
`MPPC_GET_BUFFER_SIZE`	Palauttaa puskuriin tallennettujen näytteiden lukumäärän.
`MPPC_GET_TIMER_VALUE`	Palauttaa tämänhetkisen ajastimen arvon mikrosekunteina.
`MPPC_GET_SIGNAL_STATE`	Palauttaa signaalin tilan, joka on 0 tai 1.

Liitteessä B on MPPC:n otsikkotiedosto, josta käy ilmi muun muassa *ioctl*-komentojen numerointi. Liitteessä C on ajurin otsikkotiedosto.

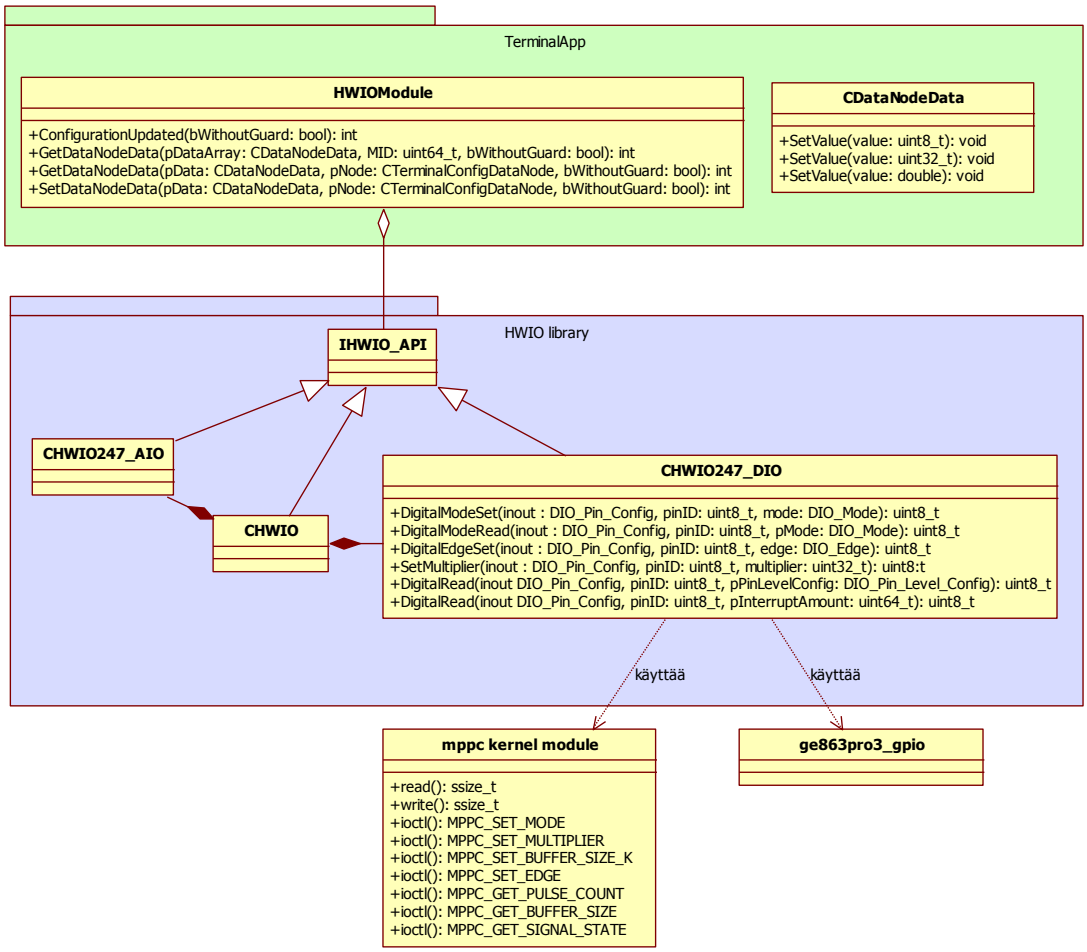
6.2 Testiohjelma

Testausta varten kirjoitettiin C-kielinen testiohjelma, jolla on mahdollista testata laiteajurin kaikkia ominaisuuksia. Testiohjelmalla voidaan tulostaa kaikki sen lukemat arvot, tai keskiarvon luetun puskurin arvoista. Testiohjelman pääfunktion koodilistaus on liitteessä D. Ohjelma tulostaa *-h* -parametrilla seuraavan aputekstin, josta käy ilmi sen toiminnot:

```
# ./test_mppc -h
Usage: test_mppc [arguments]
List of arguments:
  -a           Calculate average
  -b  [size]   Buffer size (max 1023)
  -B  [size]   Buffer size in kilos (max 64)
  -d           Restore default configuration
  -e  [enum]   Set edge (0=none, 1=rising, 2=falling, 3=both)
  -g  [value]  Use general-purpose backup registers (GPBR) 0=off, 1=on
  -h           Print this help text
  -m  [value]  Frequency multiplier (max 1024)
  -M  [mode]   Set mode (1=pulse count, 2=frequency)
  -p  [time]   Set poll time (ms)
  -t           Query timer value
  -w  [value]  Write pulse count value and exit program
  -z           Set pulse count to zero after each read
```

6.3 WRM:n terminaalisovellus

WRM:n terminaalisovellus koostuu pääohjelmasta TerminalApp ja moduuleista, joita ladataan käyttöön tarpeen mukaan. Moduulit käyttävät niihin liittyviä ohjelmistokirjastoja. Toimiakseen TerminalApp:ssa MPPC-ajuri piti integroida HWIO-kirjastoon (hardware I/O), joka toimii analogisten ja digitaalisten I/O:en rajapintana.

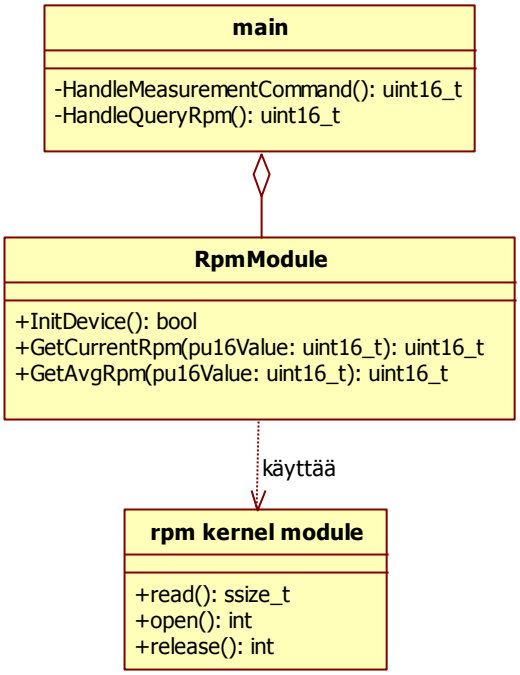


Kuva 6.2. *TerminalApp-sovelluksen luokkakaavio pulssilaskurin osalta*

Kuvassa 6.2 on terminaalisovelluksen luokkakaavio niiltä osin, kuin se koskee pulssilaskuria. *IHWIO_API* on rajapintaluokka, josta on periytetty analogista ja digitaalista puolta käsittelevät moduulit *CHWIO247_AIO* ja *CHWIO247_DIO*. Laiteajuri *ge863pro3_gpio* on Telit:n valmistama, ja se huolehtii tavallisista digitaalisten I/O:en toiminnoista [19]. Jos I/O:a käytetään pelkän tilan lukemiseen, käytetään tätä ajuria, ja pulssilaskennassa käytetään MPPC-ajuria.

6.4 Moventas Gears:n terminaalisovellus

Moventas Gears:n sovellusta varten tehty terminaalisovellus koostuu pääohjelmasta ja muutamasta moduulista, jotka huolehtivat muun muassa kommunikoinnista. Luokka *RpmModule* huolehtii kierrosnopeusmittaukseen liittyvistä asioista. Näihin kuuluu kierrosnopeuden lukeminen säännöllisesti ajurilta ja keskiarvon laskenta. Lukeminen suoritetaan oletusarvoisesti 100 millisekunnin välein ja se suoritetaan omassa säikeessään. Keskiarvo lasketaan siinä vaiheessa, kun se luetaan luokalta ja lähetetään PC-sovellukselle.



Kuva 6.3. *Moventas Gears:n terminaalisovelluksen luokkakaavio*

Kuvassa 6.3 on sovelluksen luokkakaavio. Funktioista on esitetty ne, jotka koskevat kierrosnopeuden mittaamista.

7 REAALIAIKAISUUSMITTAUKSET

Reaaliaikaisuutta mitattiin sekä säikeistetyllä keskeytyskäsittelijällä että normaalilla keskeytyskäsittelijällä. Mittauksia tehtiin eri taajuuksilla ajurin käyttöalueelta, joka on 0-50000Hz. Mittaukset suoritettiin syöttämällä funktiogeneraattorilla kanttiaaltosignaalia pulssilaskurin sisääntulonastaan ja mittaamalla ulostuloja oskilloskoopilla. Mittauksissa käytetty oskilloskooppi on malliltaan Agilent MSOX2024A, joka sisältää myös funktiogeneraattorin [17]. Mittauksia varten ajurista käännettiin erityinen testiversio, jossa laitteen ulostulot on asetettu muuttumaan tietyissä kohdissa keskeytyskäsittelijän koodin suoritusta.

Altenbergilla oli samankaltainen mittausjärjestely ARM-järjestelmän latenssimittauksissa [1]. Funktiogeneraattori oli kytketty sisääntulonastaan, jonka nouseva reuna generoi keskeytyksen. Keskeytyskäsittelijän alussa vaihdetaan ulostulon tilaa, jota mitataan oskilloskoopilla. Toinen testissä käytetty MPU oli ATSAM9263, jossa on sama ARM926-suoritin kuin WRM:n käyttämässä ATSAM9260:ssä. Eroa näiden kahden välillä on oheislaitteiden määrässä, suorittimen maksimikellotaajuudessa ja integroidun välimuistin määrässä. Altenbergin mittauksessa suorittimen kellotaajuus oli 180MHz ja WRM:n mittauksessa 200MHz. Referenssiksi otettiin myös Franken vastaavankaltainen mittaus, jossa alustana oli PC varustettuna 2,8 gigahertsin Pentium 4 -suorittimella [12].

Sekä Altenbergin että Franken mittauksissa mitattiin myös aikaa, joka kuluu kontekstin vaihtoon. Tämä on myös olennaista reaaliaikaisuuden ja suorittimen kuormituksen kannalta. Kontekstinvaihtoaikojen mittaus on kuitenkin rajattu pois tästä työstä.

7.1 Keskeytyslatenssi

Kappaleessa 5.1 esitettiin keskeytyslatenssin määritelmä. Tässä tapauksessa tarkasteltava tapahtuma on pulssin nouseva reuna, ja suoritettava koodi on keskeytyskäsittelijä. Mittausta varten keskeytyskäsittelijän koodia muokattiin siten, että heti sen alussa piirin ulostuloon generoidaan lyhyt pulssi.

Oskilloskoopilla mitattiin ulostulosignaalia suoraan järjestelmäpiirin ulostulosta, jotta pulssi havaittaisiin mittalaitteessa ilman ylimääräisiä viivetekijöitä. Ulostulojännite korotettiin oskilloskoopin matematiikkafunktiolla toiseen potenssiin, jotta pulssit erottuisivat paremmin kohinasta. Liipaisu säädettiin funktiogeneraattorin nousevalle reunalle 50 prosenttiin maksimijännitteestä. Liipaisuaika säädettiin kuvan vasempaan reunaan. Pulsseista saatiin laskettua jakauma asettamalla oskilloskooppi laskemaan liukuvaa keskiarvoa. Selkeyden vuoksi sisääntulosignaalia ei otettu mukaan kuviin. Oskilloskooppikuvien Y-akseli on suhteellinen, joten siihen ei ole merkitty arvoja. [17]

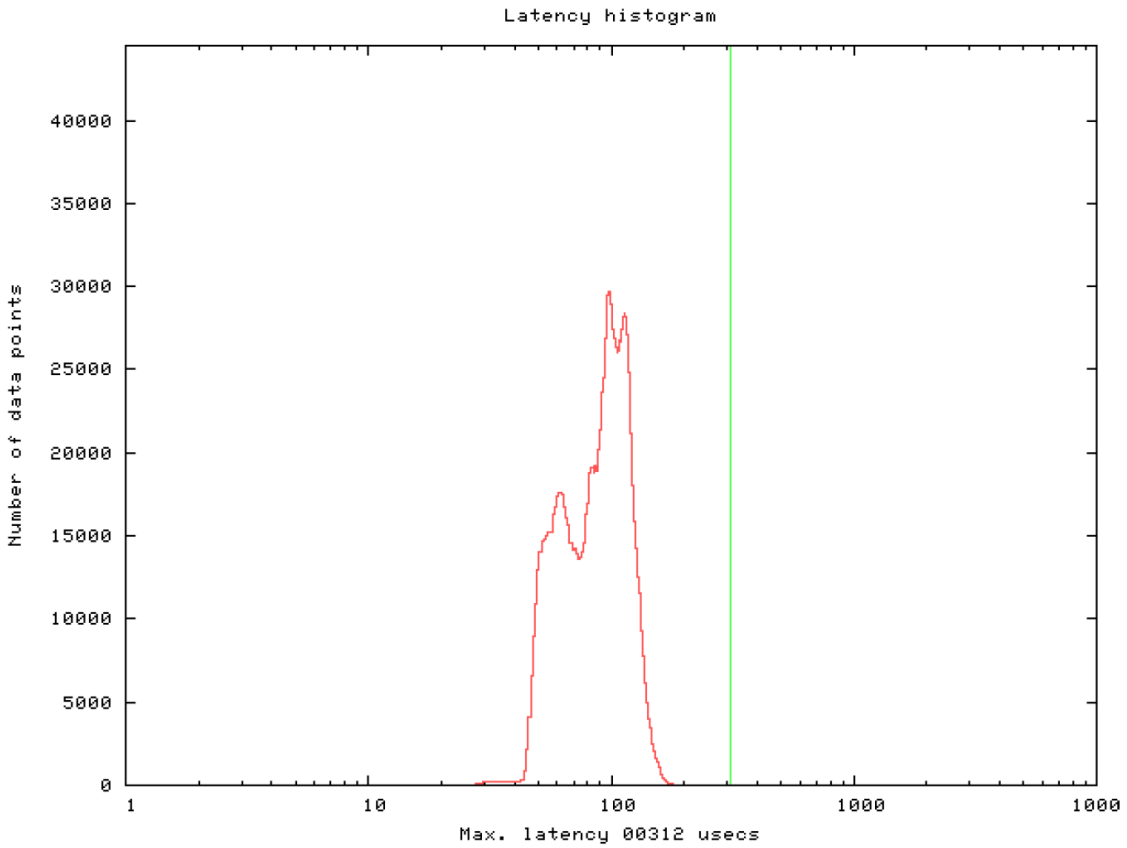
7.1.1 Liukuva keskiarvo

Kuvissa 7.2 ja 7.3 on mitattuja liukuvia keskiarvoja eri sisääntulosignaalin taajuuksilla. Kuvissa keskellä näkyvä piikki 10kHz taajuudella on häiriöpulssi, joka syntyy sisääntulosignaalin muutoksesta ylhäältä alas, sillä kyseisellä taajuudella yksi jakso on oskilloskoopin näytön levyinen.

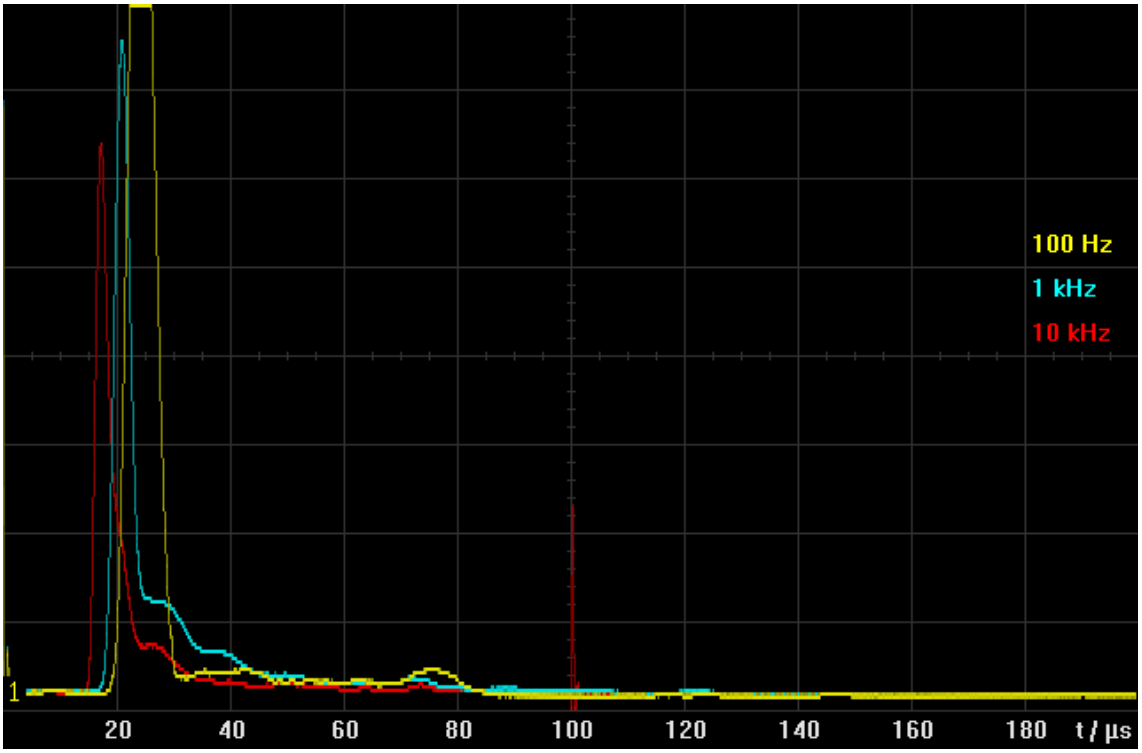
Kuten oli odotettavissa, keskeytyslatenssi on keskimäärin selvästi suurempi säiettä käytettäessä kuin normaalilla keskeytyskäsittelijällä. Jos mitataan eroa jakauman piikin korkeimmasta kohdasta, on aikaero keskeytyskäsittelijätapojen välillä 10-50µs. Säikeellä myös hajonta oli selvästi suurempaa, kun taas normaalilla käsittelijällä jakauman piikit ovat korkeita ja kapeita.

Mielenkiintoista tuloksissa on se, että sisääntulosignaalin taajuuden kasvaessa keskimääräinen latenssi pieneni. Tämä selittyy sillä, että välimuisti pitää huuhdella harvemmin, ja välimuistin osumatarkkuus on parempi. Tämä johtuu taas siitä, että keskeytyskäsittelijän koodi on todennäköisemmin välimuistissa, kun sen edellisestä käyttökerrasta on vähemmän aikaa [22]. Altenberg toteaa tekstissään, että ATSAM9263:n välimuistin huuhtelu voi viedä aikaa pahimmillaan jopa 250 mikrosekuntia [1].

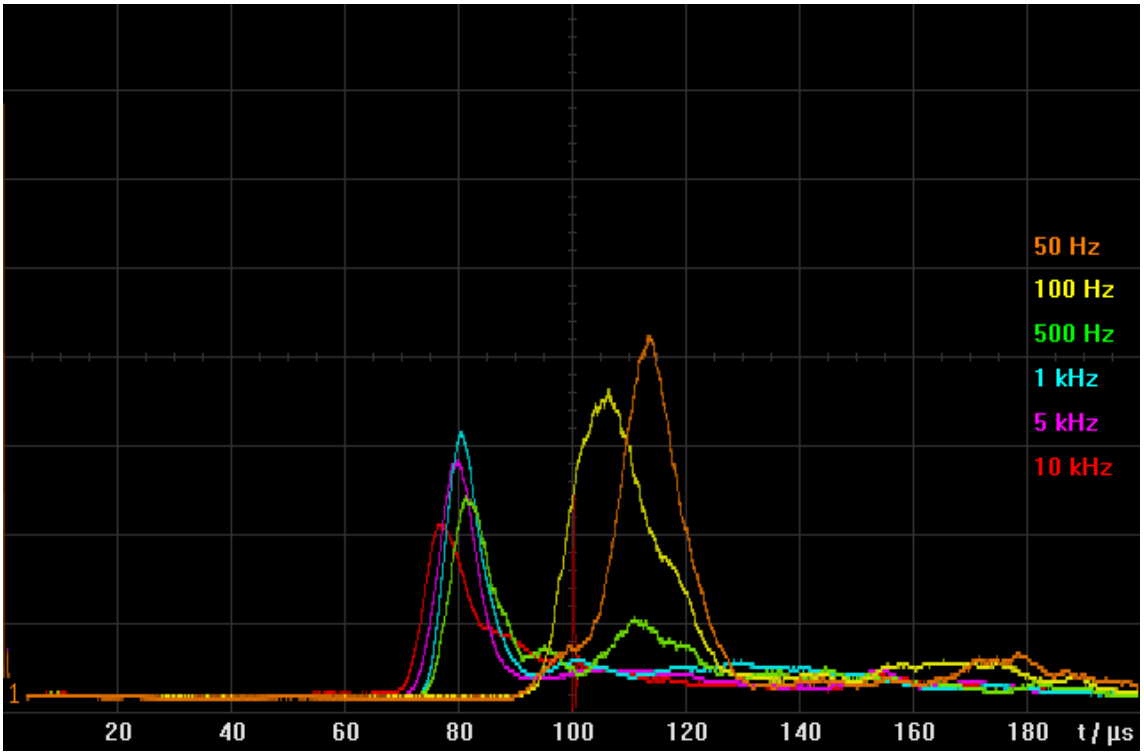
Kuvassa 7.1 on Altenbergin testin histogrammi keskeytyslatensseista 1000Hz sisääntulosignaalilla. Siinä suurin osa latensseista sijoittuu aikavälille 50-150µs ja piikin huippu on 100µs tuntumassa. Franken mittauksissa hajonta asettui kuvien 7.4 ja 7.5 mukaisesti väleille 7-11µs ja 8-17µs. Kun otetaan huomioon 14-kertainen kellotaajuus suhteessa WRM 247:n kellotaajuuteen, ovat tulokset samaa suuruusluokkaa.



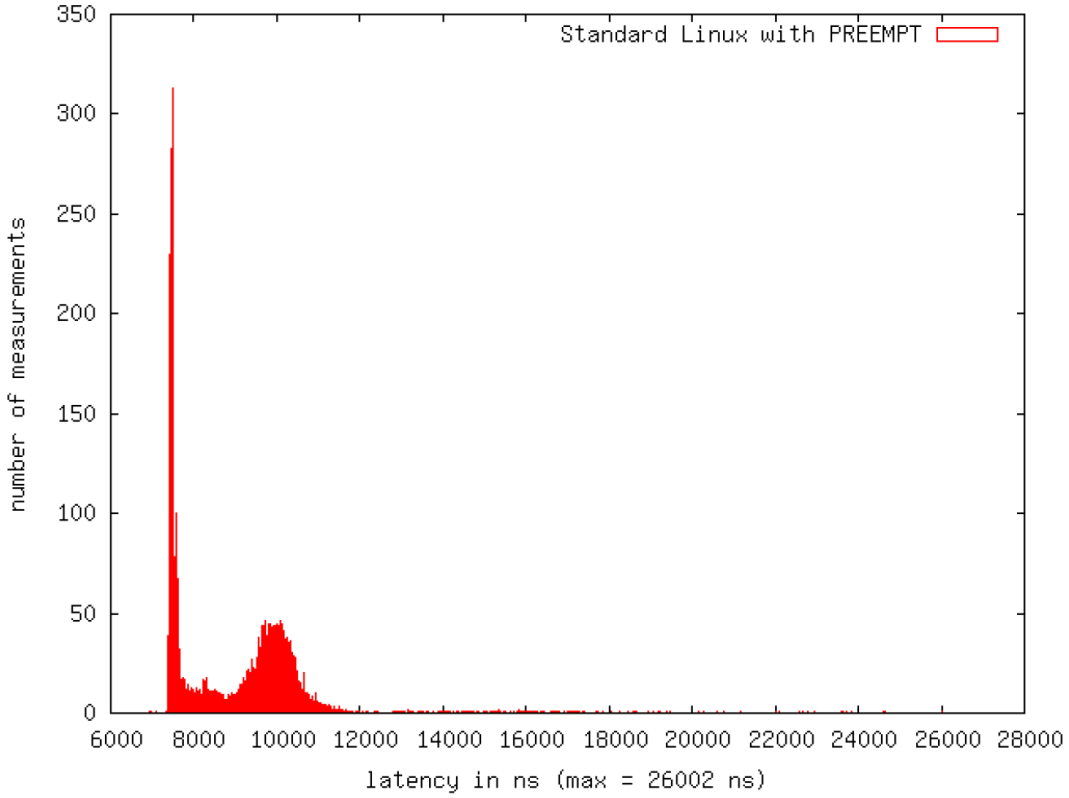
Kuva 7.1. *Altenbergin testin histogrammi [1]*



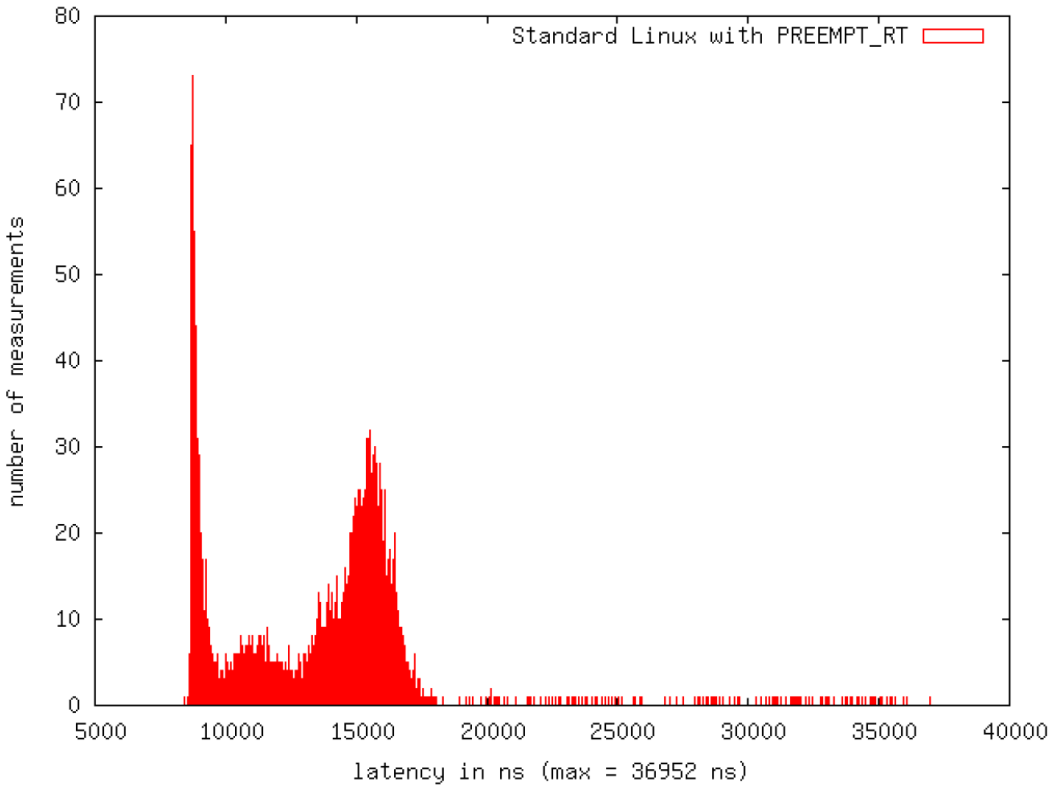
Kuva 7.2. *Keskeytyslatenssijakauma - ISR*



Kuva 7.3. *Keskeytyslatenssin jakauma - säie*



Kuva 7.4. *Franken testin histogrammi ei-reaaliaikaisella ytimellä [12]*

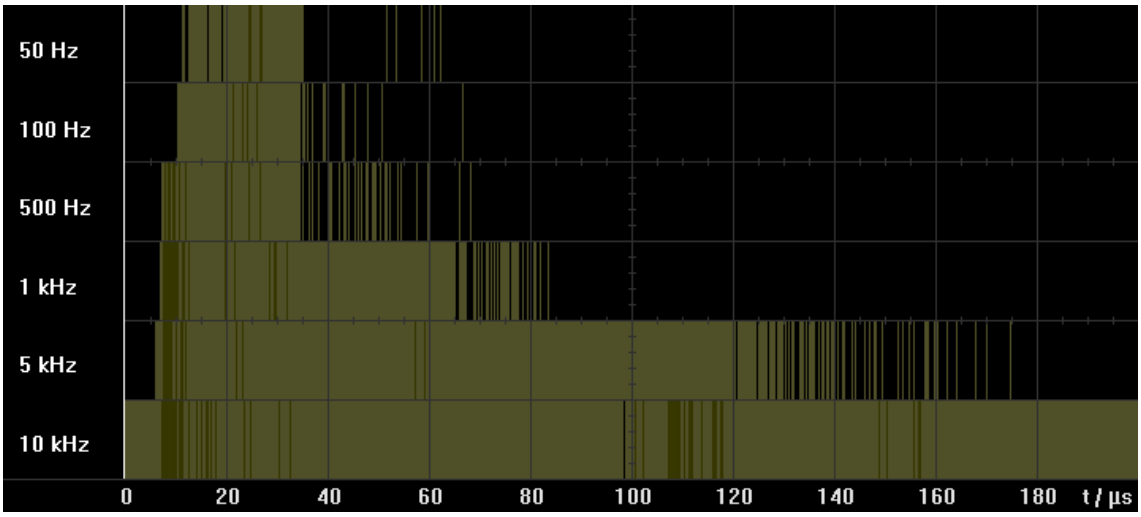


Kuva 7.5. *Franken testin histogrammi reaaliaikaytimellä [12]*

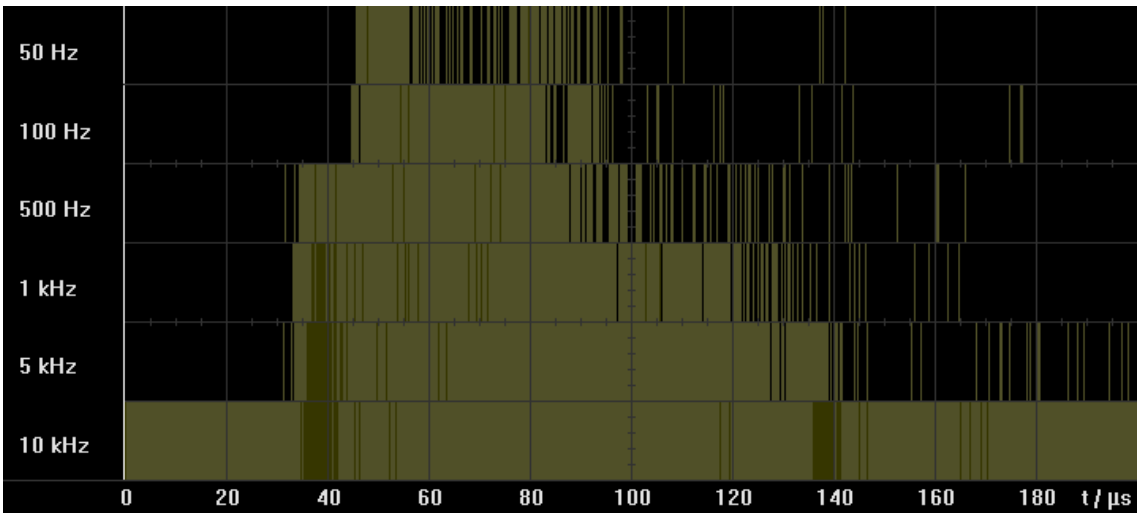
Sama ilmiö on havaittavissa WRM 247:n ja PC:n tapauksessa; keskeytyslatenssi on suurempi, kun keskeytyskäsittely tehdään säikeessä. Tämä on odotettavissa, sillä säikeen käynnistäminen vie kuitenkin aikansa. Kuten kappaleessa 4.2 todettiin, reaaliaikaisuus ei tarkoita välttämättä pienintä mahdollista latenssia.

7.1.2 Huonoin tapaus

Huonoin tapaus (worst case) mitattiin samalla tavalla kuin liukuva keskiarvo, mutta oskilloskooppi asetettiin keskiarvoistuksen sijaan persistence-tilaan [17], jolloin näyttöä ei tyhjennetä liipaisujen välissä, ja kaikki mittaukset jäävät näkyviin. Kuvasta nähdään myös paras eli lyhyin tapaus, mutta yleensä ollaan kiinnostuneempia huonoimmasta tapauksesta, sillä se määrittelee, minkälaiset suorituskykyarvot järjestelmälle voidaan luvata.



Kuva 7.6. *Kesketyslatensseja - ISR*



Kuva 7.7*. Keskeytyslatensseja - säie*

Kuvissa 7.6 ja 7.7 on yhdistetty mittaustuloksia eri taajuuksilla. Tuloksista nähdään, että taajuutta nostaessa sekä huonoin tapaus että jitter kasvavat. Toisaalta myös paras tapaus paranee taajuuden kasvaessa. Osaltaan tuloksiin vaikuttaa se, että mittausaika oli sama kaikilla taajuuksilla. Tällöin tapahtumia on mitattu esimerkiksi 200-kertainen

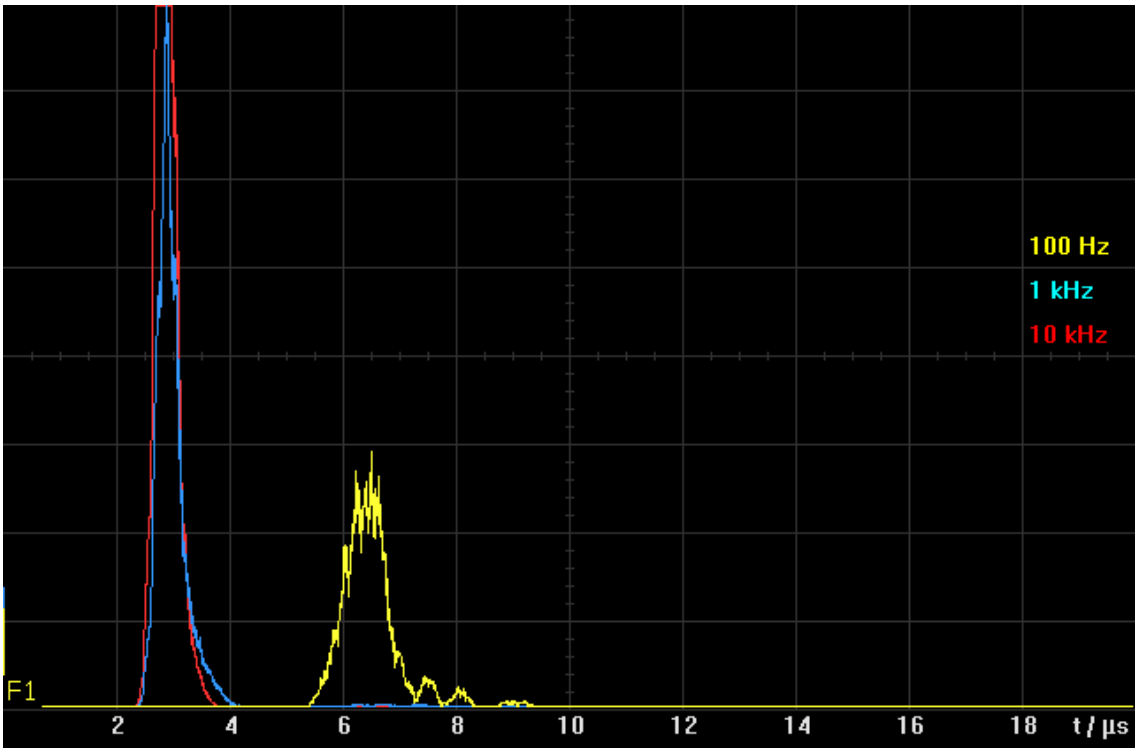
määrä 10 kilohertsillä verrattuna 50 hertsiin. Tästä syystä pitkiä latensseja on ehtinyt tulemaan enemmän korkeilla taajuuksilla kuin matalilla.

Huonoimmaksi tapaukseksi mitattiin kilohertsin signaalilla noin 84µs normaalilla keskeytyksellä ja noin 165µs säikeellä. Altenberg mittasi samalla taajuudella pitkän aikavälin huonoimmaksi tapaukseksi 248µs [12].

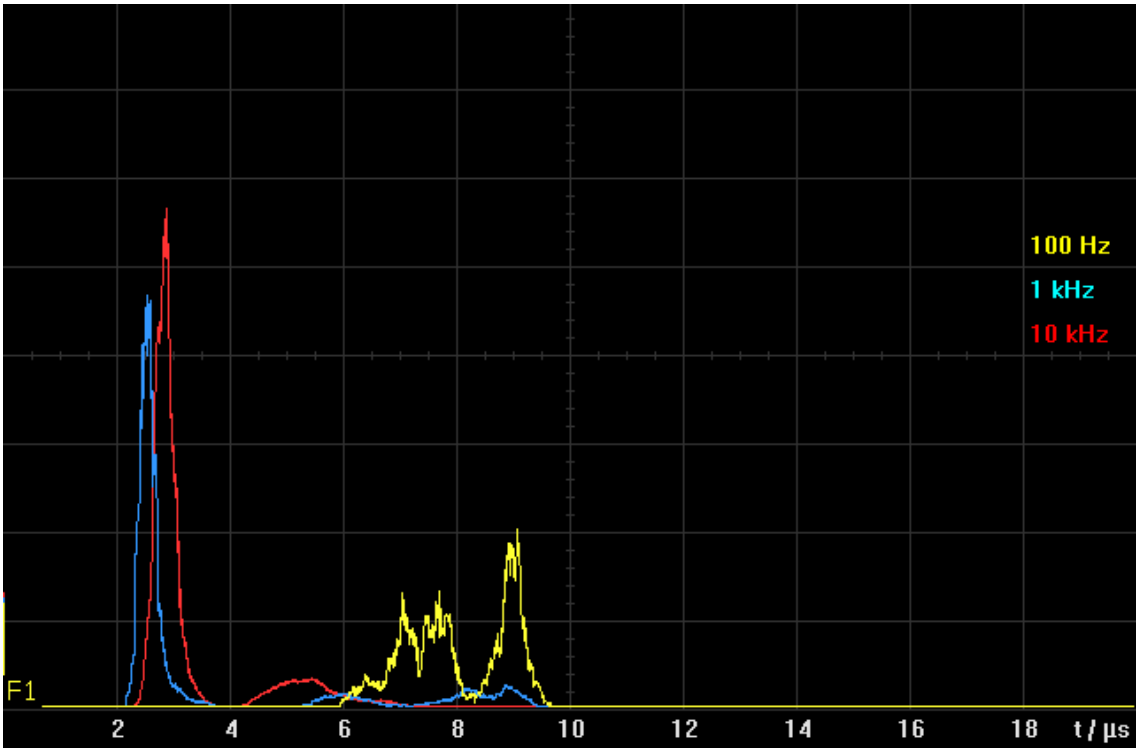
7.2 Keskeytyskäsittelijän pituus

Keskeytyskäsittelijän pituus mitattiin siten, että keskeytyskäsittelijän alussa lähetettiin pulssi yhteen ulostuloon, ja keskeytyskäsittelijän lopussa toiseen ulostuloon. Tässä ei huomioida sitä aikaa, mikä käyttöjärjestelmässä kuluu kontekstin vaihtoon, mutta testi antaa kuitenkin informaatiota keskeytyksen aiheuttamasta suoritinkuormasta.

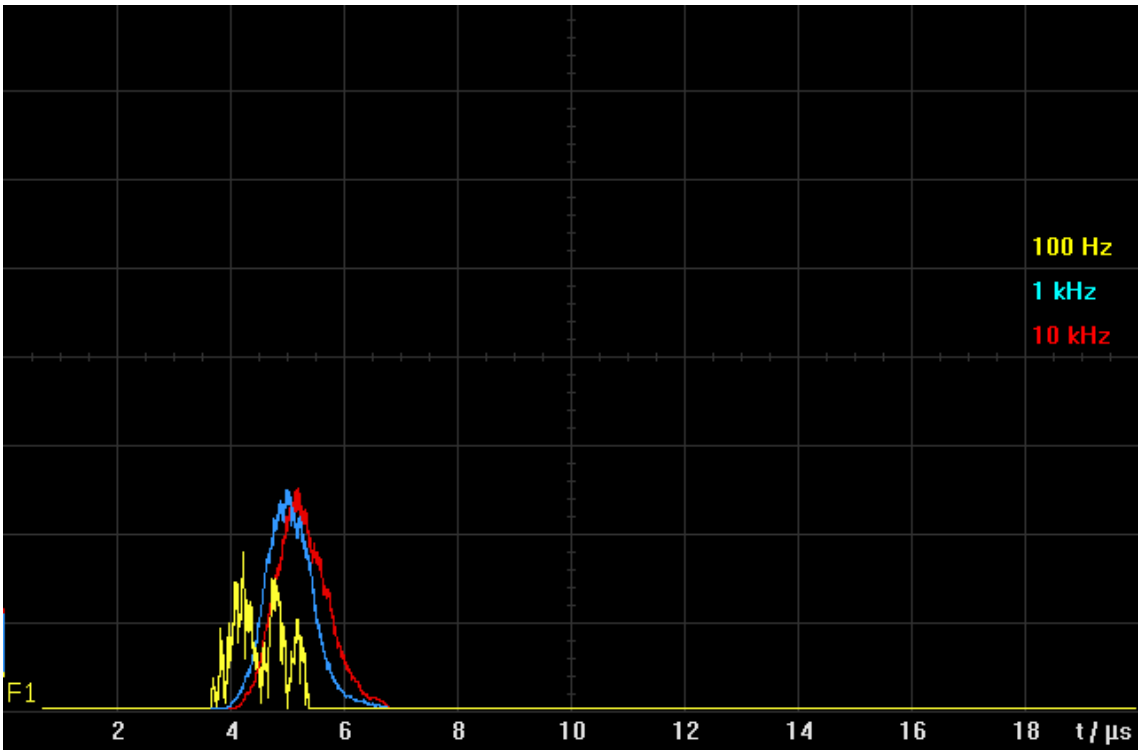
Kuvissa 7.8, 7.9, 7.10 ja 7.11 on oskilloskoopilla mitattu keskeytyskäsittelijän pituuden jakauma eri taajuuksilla. Kuvat 7.8 ja 7.10 on mitattu ilman ylimääräistä kuormaa, ja muut kaksi on mitattu luomalla samanaikaisesti kuormaa. Normaali keskeytyskäsittelijä (ISR) on odotusten mukaan säiettä nopeampi. Tämä selittyy sillä, että vuorontaja ei voi keskeyttää sitä, vaan ainoastaan korkeamman prioriteetin keskeytys voi keskeyttää sen [1]. Toisaalta säikeessä oleva keskeytys häiritsee vähemmän muuta järjestelmää, sillä se voidaan keskeyttää ja antaa suoritinaikaa välissä muille prosesseille.



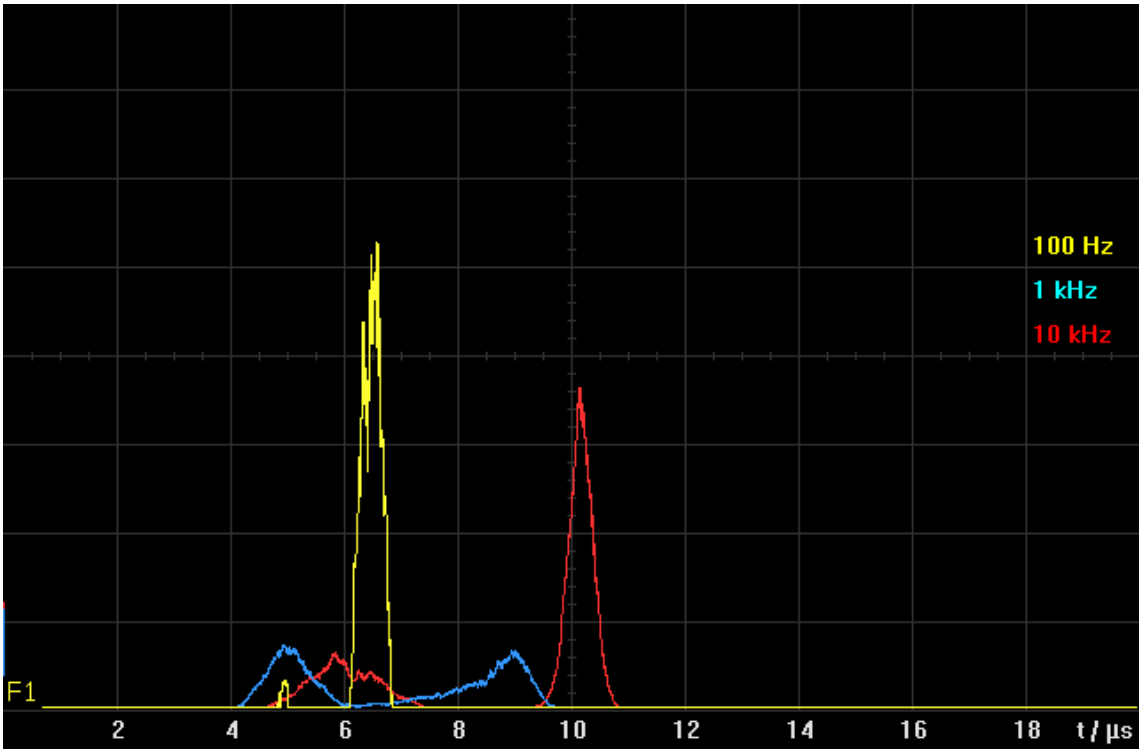
Kuva 7.8. *Keskeytyskäsittelijän pituusjakauma - ISR, kuormittamaton*



Kuva 7.9. *Keskeytyskäsittelijän pituusjakauma - ISR, kuormitettu*



Kuva 7.10. *Keskeytyskäsittelijän pituusjakauma - säie, kuormittamaton*



Kuva 7.11. *Keskeytyskäsittelijän pituusjakauma - säie, kuormitettu*

Kuormitettuna keskeytyskäsittelijän pituuden keskiarvo kasvaa. Säikeellä ero on suurempi etenkin korkeammilla taajuuksilla, sillä säie voidaan irrottaa kesken suorituksen.

7.3 Suurin toimintataajuus

Funktiogeneraattorin avulla testattiin myös suurin sisääntulosignaalin taajuus, jolla järjestelmä vielä toimii kaatumatta. Kun signaalin taajuus nousee liian korkeaksi, kuluu kaikki suoritinaika keskeytysten käsittelyyn ja kontekstin vaihtoon, jolloin vahtikoira-ajastin (watchdog timer) laukeaa ja käynnistää järjestelmän uudelleen [3].

Suurin toimintataajuus testattiin siten, että WRM 247:ssä suoritettiin samalla normaalisti muuta sen toimintaan liittyvää ohjelmistoa kuten TerminalApp:a. Säikeistettyä keskeytyskäsittelijää käytettäessä ongelmia alkoi esiintyä jo 11 kilohertsin taajuudella. Laite ei enää selviytynyt käynnistyksellä tätä suuremmilla taajuuksilla. Jos sisääntulosignaali kytkettiin vasta, kun järjestelmä oli kokonaan käynnistynyt, laukesi vahtikoira 15 kilohertsin signaalilla. Normaalia keskeytyskäsittelijää käytettäessä vahtikoira laukesi vasta 85 kilohertsin signaalilla.

Taulukossa 7.1 on mitattu Linuxin *top*-ohjelmalla [33] säikeiden suoritinkäyttöä eri sisääntulosignaalin taajuuksilla. Taulukon arvot ovat keskiarvoja minuutin jaksolta. Säikeistetyllä keskeytyskäsittelijällä voidaan tarkastella myös kesketyskäsittelijän suoritinkäyttöä, mitä ei voida tehdä normaalilla kesketyskäsittelijällä. Taulukossa on IRQ-säikeen lisäksi TerminalApp-säikeen suoritinkäyttö kyseisillä taajuuksilla. Tämä on mitattu sekä normaalilla (oikeanpuolimmaisin sarake) että säikeistetyllä kesketyskäsittelijällä. 14 kilohertsin signaalilla IRQ:n ja TerminalApp:n yhteelaskettu suoritinkäyttö on jo 87%. Tästä syystä on täysin ymmärrettävää, että järjestelmä kaatuu 15 kilohertsin signaalilla.

Taulukko 7.1. *Suorittimen käyttöaste eri sisääntulosignaalin taajuuksilla*

Taajuus (kHz)	IRQ - säie (%)	TerminalApp - säie (%)	TerminalApp - ISR (%)
10	29	16	5
12	42	30	7
14	55	32	10

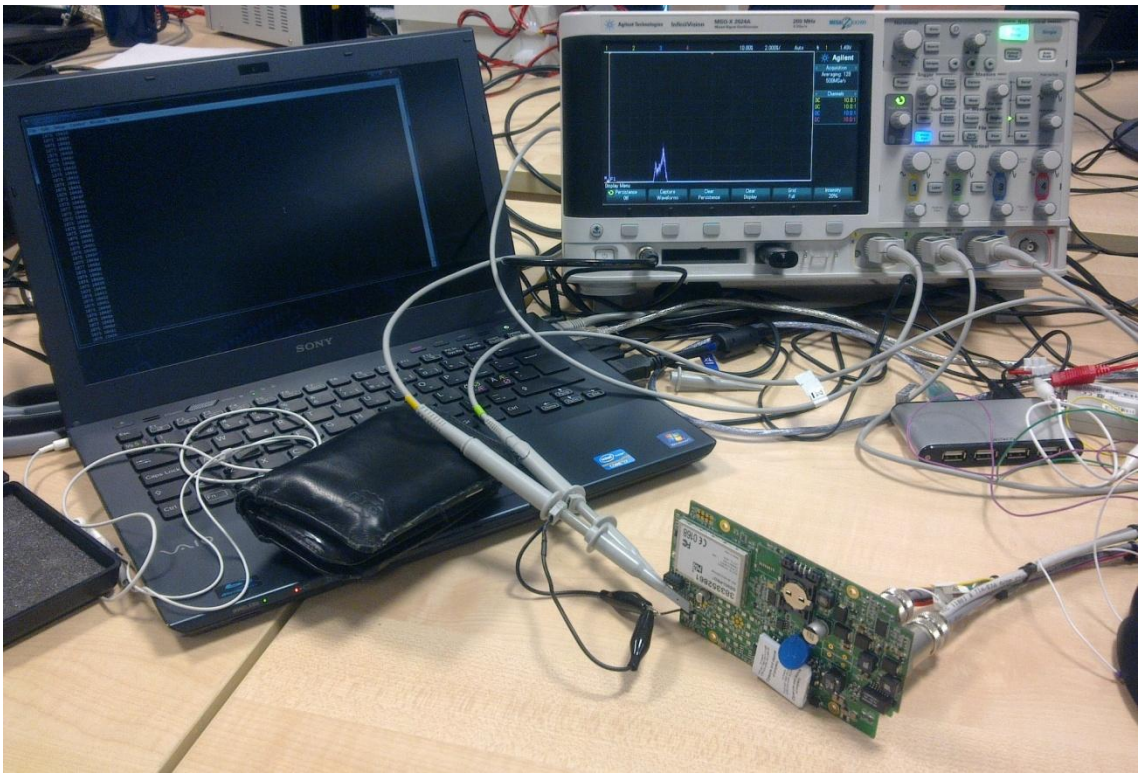
Kaatuminen suurella sisääntulotaajuudella ehkäistiin siten, että ajuriin ohjelmoitiin toiminnallisuus, joka kytkee pulssinreunakeskeytyksen pois, jos sisääntulosignaali on ollut liian korkea sekunnin ajan. Tällöin muut prosessit saavat välillä suoritinaikaa ja vahtikoira ei pääse laukeamaan. Sekunnin päästä keskeytys kytketään takaisin päälle, ja jos liian korkeataajuiset pulssit jatkuvat, kytketään keskeytys taas pois sekunnin kuluttua. Rajataajuudeksi valittiin 56kHz, joka on 85 kilohertsin maksimitaajuutta selvästi alhaisempi.

Kuva 7.12. *Taajuuspyyhkäisy 0-200kHz*

Edellä mainitun toiminnon ansiosta suurempiakin taajuuksia pystytään mittaamaan maksimissaan sekunnin pätkissä. Absoluuttista maksimitaajuutta testattiin nostamalla taajuutta hitaasti nollasta kahteensataan kilohertsiin. Kuvassa 7.12 on kuvakaappaus WRM:n käyttöliittymän kuvaajasta, jossa on testin tulokset. X-akselilla on kellonaika ja Y-akselilla on taajuus hertseinä. Kuvasta nähdään, että taajuuden noustessa yli 56 kilohertsin, muuttuu kuvaaja porrasmaiseksi, sillä keskeytys on jaksoittain pois päältä. Yli 120 kilohertsin taajuudella pulsseja alkoi selvästi hukkua siinä määrin, että taajuusmittaus muuttui varsin epäluotettavaksi. Suurin oikein mitattu taajuus oli hieman alle 140 kilohertsiä.

8 TARKKUUSMITTAUKSET

Tarkkuutta mitattiin funktiogeneraattorin avulla. Vakiotaajuudella testatessa käytettiin Agilent MSOX2024A:n funktiogeneraattoria [17] ja muuttuvalla taajuudella testatessa PicoScope 2205 MSO -oskilloskoopin funktiogeneraattoria [24]. Ohjelman mittaamia arvoja vertailtiin funktiogeneraattorin taajuuteen tai siitä laskettuihin arvoihin. Kuormitetuissa testeissä kuormaa synnytettiin esimerkiksi luomalla WRM 247:n ja PC:n välille verkkoliikennettä.



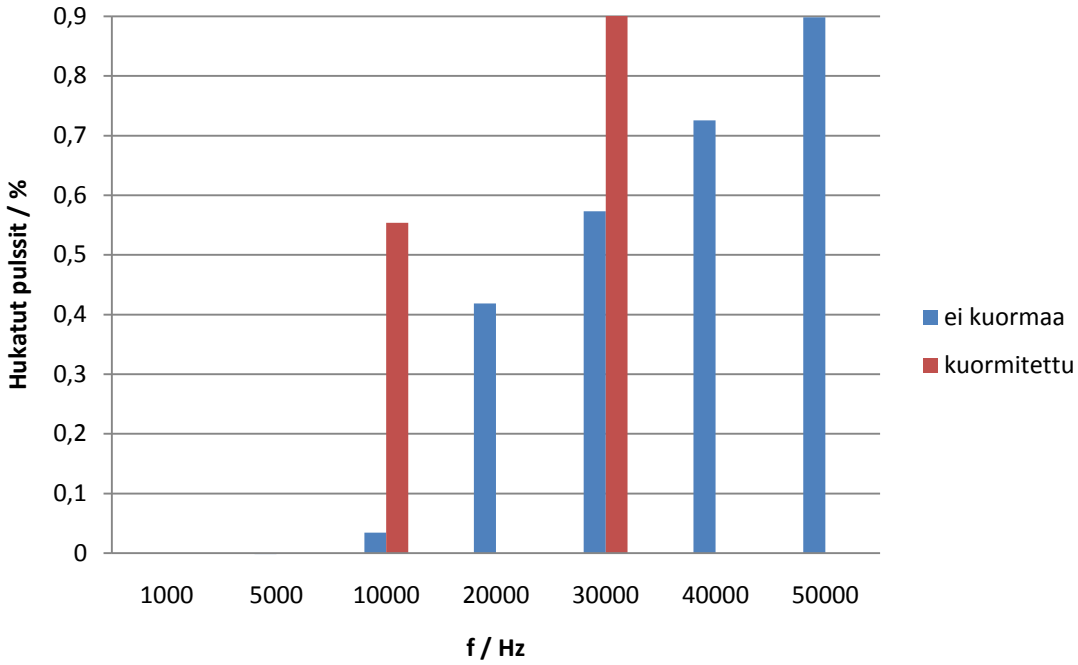
Kuva 8.1. *Testijärjestely Agilent MSOX2024A -oskilloskoopin kanssa*

Kuvassa 8.1 on testijärjestely MSOX2024A:n kanssa. Etualalla on koteloimaton WRM 247 -terminaali. Oskilloskoopin mittapäät on kytketty digitaalisten ulostulojen kytkentätransistorien hiloille, joihin signaali tulee suoraan järjestelmäpiiriltä. PC:llä on auki terminaali, jolla käytetään WRM 247:n konsolia sarjayhteyden kautta.

8.1 Hukatut pulssit

Pulssilaskennan tarkkuutta mitattiin siten, että funktiogeneraattori asetettiin tuottamaan vakiotaajuista kanttiaaltoa, ja mitattiin laiteajurin havaitsemien pulssien määrää. Testiohjelma tulosti pulssilukeman kymmenen sekunnin välein. Pulssilukema kirjattiin testin alussa ja 60 minuutin kuluttua testin alkamisesta. Näiden erotusta verrattiin pulssilukeman laskennalliseen arvoon, joka saatiin kaavasta $n = f * t$, missä f on funktiogeneraattorin taajuus ja, t on mittauksen aikaväli. Mittaus suoritettiin kuormittamattomalla järjestelmällä seitsemällä eri taajuudella ja kuormitetulla järjestelmällä kahdella.

Mittaustulokset on esitetty kuvassa 8.2, jossa pystyakselilla on hukattujen pulssien lukumäärä suhteessa mitattujen pulssien määrään prosentteina. Funktiogeneraattorin virhe on taajuudesta riippuen 0,005 ja 0,010 prosentin välillä [17]. 1000 ja 5000 hertsin signaalilla hukattujen pulssien suhteellinen määrä oli pienempi kuin funktiogeneraattorin virhe [17], mistä voidaan tehdä johtopäätös, että näillä taajuuksilla pulsseja ei hukattu. Kymmentä kilohertsiä suuremmilla taajuuksilla virhe alkoi kasvaa. Kuitenkin vielä 50 kilohertsin taajuudella virhe oli alle prosentin kuormittamattomana. Kuormitettuna virhe oli muutaman prosenttiyksikön kymmenesosan suurempi.



***Kuva 8.2**. Pulssilaskennan tarkkuus*

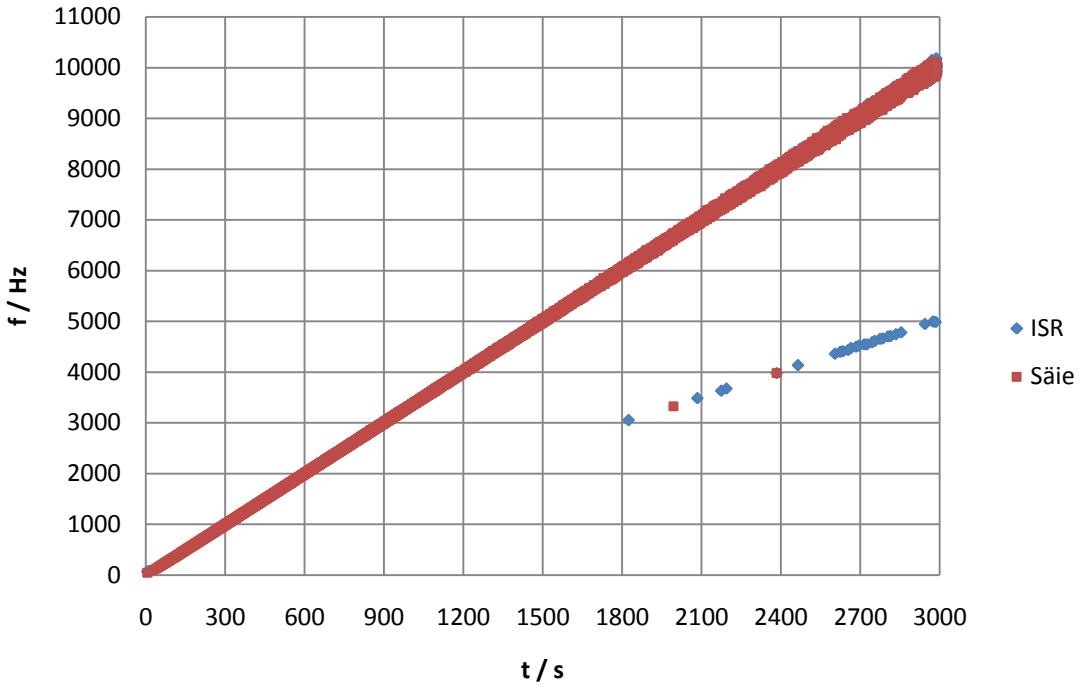
Kun tarkastellaan kuvan 7.6 tuloksia, nähdään että viiden kilohertsin taajuudella keskeytyslatenssin huonoin tapaus on pienempi kuin jaksonaika, kun taas kymmenen kilohertsin taajuudella se on jo jaksonaikaa suurempi. Tämä selittää sen, miksi alle 10 kilohertsillä pulsseja ei käytännössä hukata, mutta suuremmilla taajuuksilla hukataan. Kaikkia keskeytyskäsittelijöitä ei ehditä suorittamaan jos keskeytyslatenssi on jaksonaikaa pidempi.

8.2 Taajuuspyyhkäisytestit

Tarkkuutta testattiin myös muuttuvalla taajuudella. Funktiogeneraattori asetettiin pyyhkäisemään lineaarisesti asetetun taajuusalueen alapäästä yläpäähän asetetussa ajassa. MPPC-ajuri asetettiin mittaamaan taajuutta 64 kilonäytteen puskurilla. Parametrit asetettiin siten, että näytteet koko taajuuspyyhkäisyn ajalta mahtuvat puskuriin. Tuloksia tarkastellessa on hyvä ottaa huomioon se, että näytteistystaajuus on sama kuin signaalintaajuus. Tämä vaikuttaa osaltaan myös siihen, että suuremmilla taajuuksilla syntyy enemmän virhearvoja.

8.2.1 Taajuuspyyhkäisy 0-10000Hz

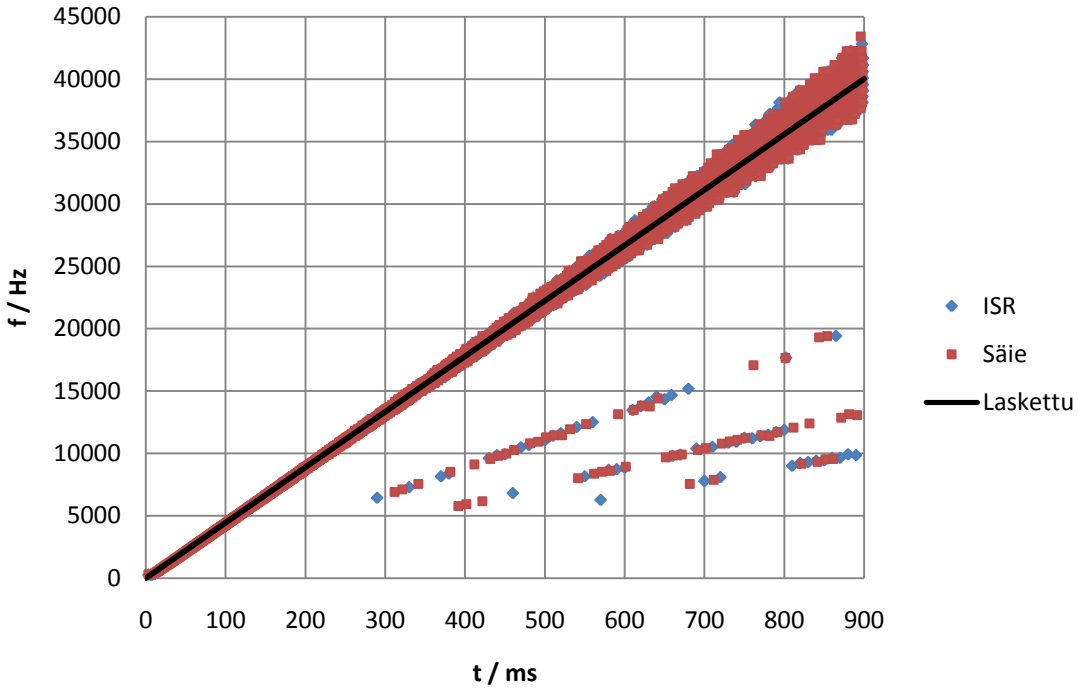
Funktiogeneraattori asetettiin pyyhkäisemään taajuusalue nollasta 10000 hertsiin kolmen sekunnin aikana. Testi tehtiin sekä kuormitettuna että ilman kuormaa. Kuormitetun testin tulokset näkyvät kuvassa 8.3. Yli 3000 hertsin taajuudella näkyy näytteitä, joissa taajuus on virheellisesti mitattu puoleksi todellisesta. Tämä johtuu siitä, että yksi pulssi on hukattu välistä. Kuvasta nähdään, että säikeistetyllä keskeytyskäsittelijällä pulsseja hukataan vähemmän. Kuormittamattomassa testissä pulsseja ei hukattu, joten sen kuva on jätetty pois vähäisen informaatioarvon vuoksi. Hukattuja pulsseja lukuun ottamatta mittaustarkkuus pysyy hyvänä koko pyyhkäisyalueella. Viiva pysyy kauttaaltaan suurin piirtein yhtä leveänä.



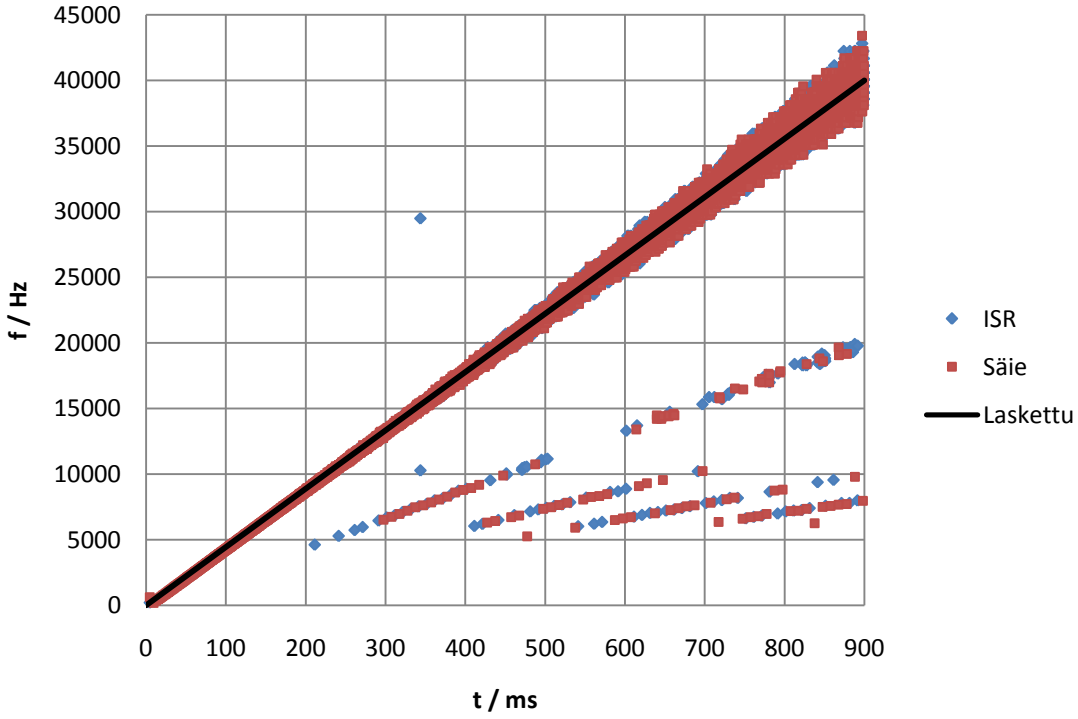
Kuva 8.3. *Taajuuspyyhkäisy 0-10000Hz*

8.2.2 Taajuuspyyhkäisy 0-40000Hz

Funktiogeneraattori asetettiin pyyhkäisemään taajuusalue nollasta 40000 hertsiin 900 millisekunnin aikana. Testi suoritettiin ilman kuormitusta ja kuormitettuna. Kuormittamattoman testin tulokset ovat kuvassa 8.4 ja kuormitetun testin kuvassa 8.5.



Kuva 8.4. *Taajuuspyyhkäisy 0-40000Hz kuormittamattomana*



Kuva 8.5. *Taajuuspyyhkäisy 0-40000Hz kuormitettuna*

Perustarkkuuteen kuormitus ei juuri vaikuta, mutta hukattuja pulsseja on selvästi enemmän kuormitettuna. Lisäksi kuormitetun testin kuvaajassa näkyy 340ms paikkeilla näyte, jonka taajuus on kaksinkertainen todelliseen taajuuteen. Pulsseja on hukkunut välillä useita peräkkäisiä, tämä näkyy kuvaajassa kolmasosa-, neljäsosa-, viidesosa- ja kuudesosataajuutena. Kuten 0-10000Hz taajuuspyyhkäisyssä, on hukattuja pulsseja enemmän ISR:lla kuin säikeistetyllä keskeytyskäsittelijällä.

8.3 Hajonta

Suhteellinen keskihajonta mitattiin lukemalla RPM-laiteajurilta 100 millisekunnin välein viimeisin taajuusarvo 40 sekunnin ajalta. Testissä tutkittiin myös latenssin kompensoinnin vaikutusta mittaustarkkuuteen, joten testi suoritettiin kahdella laiteajurin versiolla, joista toisessa latenssi oli kompensoitu ja toisessa ei.

Suhteellinen poikkeama (taulukoissa *poikkeama%*) on laskettu kaavalla

$$\frac{min - maks}{\bar{x}} * 100\%, \tag{8-4}$$

missä $\bar{x}$ on näytteiden keskiarvo. Suhteellinen keskihajonta (taulukossa *keskihajonta%*) on keskihajonnan suhde keskiarvoon. Keskihajonta on laskettu Microsoft Excelin funktiolla [18], josta saadaan suhteellisen keskihajonnan kaavaksi

$$\sigma = \frac{\sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{(n-1)}}}{\bar{x}} * 100\%, \tag{8-5}$$

missä tulos on prosentteina.

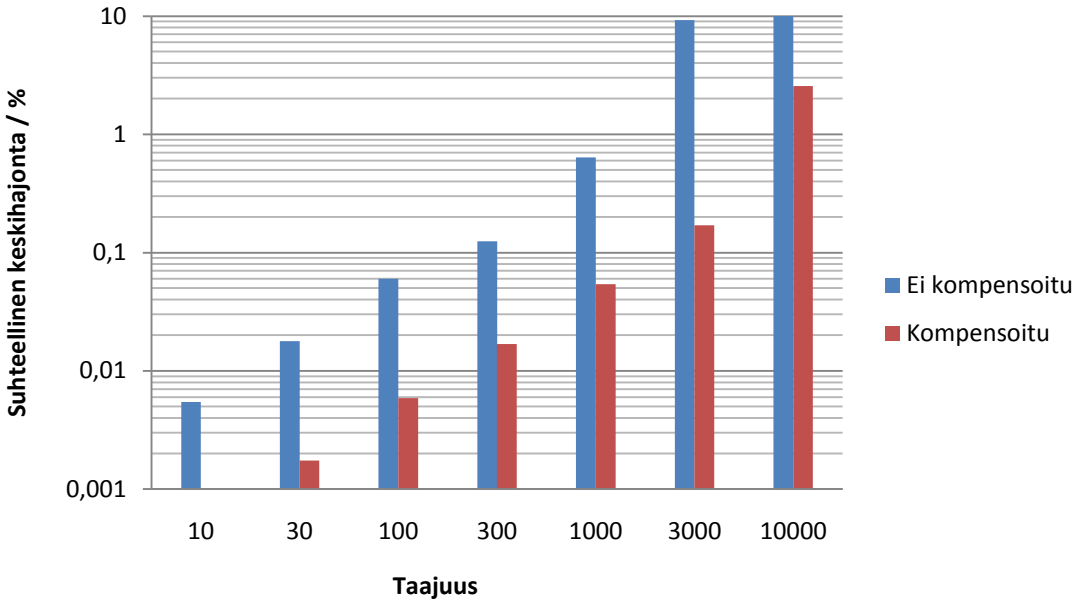
Taulukossa 8.1 on mittauksen tulokset ilman latenssin kompensointia ja taulukossa 8.2 latenssin kompensoinnin kanssa. Mittaustulokset *minimi*, *keskiarvo* ja *maksimi* ovat muodossa 1/min (rpm).

Taulukko 8.1. *Hajontamittauksen tulokset ilman latenssin kompensointia*

taajuus (hz)	10	30	100	300	1000	3000	10000
Minimi	599,9	1799,1	5986,6	17898	58302	131671	504032
Keskiarvo	600,0	1800,0	6000,2	18008	59636	173993	586372
Maksimi	600,1	1800,9	6015,0	18092	60640	332447	1065341
poikkeama%	0,03	0,10	0,47	1,1	3,9	115,4	95,73
keskihajonta%	0,005	0,018	0,06	0,12	0,64	9,24	13,56

Taulukko 8.2. *Hajontamittauksen tulokset kompensoidulla latenssilla*

taajuus (hz)	10	30	100	300	1000	3000	10000
minimi	600,0	1799,9	5999,2	17993	59904	179254	300481
keskiarvo	600,0	1800,0	6000,1	18000	60002	180014	599749
maksimi	600,0	1800,1	6001,0	18006	60077	180810	610749
poikkeama%	0,00	0,01	0,03	0,08	0,29	0,86	51,73
keskihajonta%	0,000	0,002	0,006	0,017	0,054	0,17	2,57



Kuva 8.6. *Suhteellinen keskihajonta*

Kuvassa 8.6 on havainnollistettu tuloksia logaritmista y-akselia käyttäen. Kuvasta nähdään, että suhteellinen keskihajonta kasvaa lähes logaritmisesti taajuuden funktiona lukuun ottamatta poikkeamaa 3000Hz taajuudella. Mittausten perusteella latenssin kompensointi pienentää suhteellisen keskipoikkeaman suurin piirtein kymmenesosaan. Latenssin kompensointi toisin sanoen pienentää keskeytyslatenssin jitterin vaikutusta tulokseen.

9 YHTEENVETO

Pulssilaskuri täytti Moventas Gears:n asettamat vaatimukset tarkkuuden ja suorituskyvyn suhteen. Testien mukaan taajuusmittauksessa saavutettiin vakaasti yli viisinkertainen taajuus vaadittuun 1000 hertsiin nähden. Suhteellinen keskihajonta pysyi alle puolen promillen vaaditulla taajuusalueella.

Keskeytyksissä päädyttiin käyttämään normaalia keskeytyskäsittelijää säikeen sijaan. Tähän päätyminen ei ollut ilmiselvää, sillä taajuuspyyhkäisytesteissä hukattuja pulsseja tuli tällä ratkaisulla enemmän. Kuitenkin järjestelmä toimi vakaammin normaalia keskeytystä käyttäessä, millä on suurempi painoarvo. WRM 247:ssä ei kuitenkaan ole kovia reaaliaikavaatimuksia, joten ei-reaaliaikaisen tavan käyttäminen ei häiritse järjestelmää kriittisesti. Jos useiden kymmenien tai satojen kilohertsien taajuuksia halutaan mitata vakaasti, pitää käyttää erillistä piiriä tai suoritinta, jossa on laitteistotuki pulssilaskennalle.

Taajuuden keskiarvo päädyttiin laskemaan painotettua keskiarvoa käyttäen. Laskenta ei raskaudessaan eroa juurikaan tavallisesta keskiarvosta, mutta tarkkuus on parempi tähän sovellukseen. Tämä on toki sovelluskohtaista, mutta keskiarvoalgoritmi on pienellä työllä muutettavissa.

Wapicen kannalta tutkimuksen tulokset toivat lisää tietoa ja empiirisiä tutkimustuloksia RT Patch Linuxin reaaliaikaisuudesta ja muista ominaisuuksista. Tuloksia voidaan hyödyntää niin WRM:n uutta ohjelmistoa ja laitteistoa kehitettäessä kuin muissakin vastaavanlaisissa projekteissa. Wapicella tehdään paljon ohjelmistoa sulautettuihin Linux-järjestelmiin, mihin tämä tutkimus voi tuoda hyödyllistä tietoa. Työ auttoi myös hahmottamaan laiteajurin tekemisen työmäärän laajuutta, mikä auttaa tulevien projektien budjetoinnissa.

Itselleni työ opetti etenkin Linuxin laiteajurien tekemisestä ja keskeytysmekanismin toiminnasta. Myös yleinen tuntemus Linuxin ytimestä reaaliaikapäivityksineen kasvoi merkittävästi työtä tehdessä. ARM926-suoritinarkkitehtuuriin tuli myös perehdyttyä syvemmin etenkin ajastimien osalta. Työn tekeminen muistutti myös jälleen kerran siitä, että tietotekniikassa harva asia on niin yksinkertaista toteuttaa, kuin miltä se aluksi vaikuttaa.

LÄHTEET

[1] Altenberg, J. Using Realtime Preemtion Patch on ARM CPUs. Uhldingen-Muehlhofen, Linutronix GmbH. 8 s. [viitattu 19.3.2013]. Saatavilla: http://lwn.net/images/conf/rtlws11/papers/proc/p11.pdf

[2] Anderson, M. 2009. Interrupt Threads in Linux. USA, The PTR Group, Inc. 35 s. [viitattu 18.3.2013]. Saatavilla: http://elinux.org/images/e/ef/InterruptThreads-Slides_Anderson.pdf

[3] AT91SAM ARM-based Embedded MPU, SAM9260, Atmel Corporation [datalehti, Rev. 6221K–ATARM–15-Oct-12, lokakuu 2012]. [viitattu 30.11. 2012]. Saatavilla: http://www.atmel.com/Images/doc6221.pdf

[4] AT91SAM ARM-based Embedded MPU, SAM9263, Atmel Corporation [datalehti, Rev. 6249K–ATARM–28-Jan-13, tammikuu 2013]. [viitattu 19.3.2012]. Saatavilla: http://www.atmel.com/Images/doc6178.pdf

[5] Barr, M., Massa, A. 2006. Programming Embedded Systems with C and GNU Development Tools. Sebastopol, O'Reilly Media Inc. 301 s.

[6] Bovet, D., Cesati, M. 2005. Understanding Linux Kernel. Third Edition. Sebastopol, O'Reilly Media, Inc. 923 s.

[7] Cheng, P., Yang, Y, Oelmann, B. 2011. Stator-Free RPM Sensor Using Accelerometers - A Statistical Performance Simulation by Monte Carlo Method, IEEE Sensor Journal 12/2011. Manchester, IEEE Sensor Council. 9 s

[8] Comparing Speed Sensor Technologies: Hall and VR, Cherry Electrical Products [muistio]. [viitattu 9.3.2013]. Saatavilla: http://www.cherrycorp.com/english/cherry/Hall_and_VR_speed_sensors.pdf

[9] Corbet, J., Rubini, A., Kroah-Hartman, G. 2007. Linux Device Drivers. Third Edition. Sebastopol, O'Reilly Media Inc. 640 s.

[10] Crabtree M. 2009. Industrial Flow Measurement, [diplomityö]. The University of Huddersfield. 244 s.

[11] Dilemma, Nilsson, J. 2010. Analyzing Trionic 5 with T5Suite 2.0, [käyttöohje, Rev. 1.23]. [viitattu 30.11.2012]. Saatavilla: http://trionic.mobixs.eu/Trionic%205.pdf

[12] Franke, M., A Quantitative Comparison of Realtime Linux Solutions [seminaaripaperi]. Chemnitz University of Technology, Department of Computer Science. 41 s. [viitattu 2.5.2013]. Saatavilla: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.4601&rep=rep1&type=pdf

[13] GE863-PRO^3 Embedded, Telit Communications [esite, Rev. 06.201.12]. [viitattu 30.11. 2012]. Saatavilla: http://www.telit.com/module/infopool/download.php?id=725

[14] GE863-PRO^3 Hardware User Guide, Telit Communications PLC [käyttöohje , Rev 6 -2010-09-20]. [viitattu 30.11.2012]. Saatavilla: http://www.telit.com/module/infopool/download.php?id=943

[15] Haikala, I., Järvinen, H. 2004 Käyttöjärjestelmät. 2. painos. Helsinki, Talentum Media Oy. 246 s.

[16] Inductive proximity sensor, Cylindrical threaded housing, IMF18, SICK AG [datalehti]. [viitattu 10.5.2013]. Saatavilla: https://www.mysick.com/PDF/Create.aspx?ProductID=52090&Culture=en-US

[17] InfiniiVision 2000 X-Series Oscilloscopes, Agilent Technologies [datalehti, 3.12.2012]. [viitattu 18.3.2013]. Saatavilla: http://cp.literature.agilent.com/litweb/pdf/5990-6618EN.pdf

[18] Keskihajonta [Microsoft Officen ohjesivut]. [viitattu 6.5.2013]. Saatavilla: http://office.microsoft.com/fi-fi/excel-help/keskihajonta-HP005209277.aspx

[19] Low Level Dirver APIs Application Note, Telit Comminications S.p.A. [lisätietoartikkeli, Rev 1 - 11/09/08]. [viitattu 9.5.2013]. Saatavilla: http://www.telit.com/module/infopool/download.php?id=1174

[20] The Linux Kernel Archives. [www]. [viitattu 20.3.2013]. https://www.kernel.org/

[21] LMV331 Quad General Purpose, Low Voltage, Tiny Pack Comparators, National Semiconductor [datalehti, kesäkuu 2004]. [viitattu 7.5.2013]. Saatavilla: http://www.datasheetcatalog.org/datasheet2/6/0rc20ewq7q5u7dapsewyui13h77y.pdf

[22] Love, R. 2010. Linux Kernel Development. Third Edition. USA, Pearson Education, Inc. 440 s.

[23] Mauerer, W. 2008. Professional Linux Kernel Architecture. Indianapolis, Wiley Publishing, Inc. 1337 s.

[24] PicoScope 2205 MSO USB-powered Mixed Signal Oscilloscope, Pico Technology [datalehti, Rev. MM031.en-3]. [viitattu 6.5.2013]. Saatavilla: http://www.picotech.com/document/datasheets/PicoScope2205MSO.pdf

[25] Proximity Sensors, Omron Industrial Automation [opas]. [viitattu 7.5.2013]. http://www.omron-ap.com/technical_guide/proximity_sensor/

[26] Raspberry Pi [www]. [viitattu 3.5.2013]. http://www.raspberrypi.org/

[27] Real-Time Linux Wiki [www]. [viitattu 20.3.1013]. https://rt.wiki.kernel.org/index.php/Main_Page

[28] Rostedt, S. 2009. Lockless Ring Buffer Design, Red Hat Inc. [www]. [viitattu 6.5.2013]. http://www.mjmwired.net/kernel/Documentation/trace/ring-buffer-design.txt

[29] RT Patch lataussivu, kernel.org. [viitattu 20.3.2013]. Saatavilla: https://www.kernel.org/pub/linux/kernel/projects/rt/

[30] Sähkö & Tele, 1/2013. Helsinki, Sähköinsinööriliitto ry. 42 s

[31] Tachometer Signals and Transducers, National Instruments [opas]. [viitattu 10.3.2013]. Saatavilla: http://zone.ni.com/reference/en-XX/help/372416A-01/svtconcepts/tacho_sig/

[32] Tennis C. 2007. A Peek at Computer Electronics. Raleigh, The Pragmatic Book-shelf, 250 s.

[33] top(1) – Linux man page [www]. [viitattu 10.5.2013]. Saatavilla: http://linux.die.net/man/1/top

[34] W12-3 Photoelectronic Sensors, SICK AG [datalehti, 2010-03-03]. [viitattu 3.5.2013]. Saatavilla: https://mysick.com/saqqara/im0024545.pdf

[35] WRM 247, Wapice Oy [datalehti]. [viitattu 30.11.2012] Saatavilla: http://www.wrm.fi/files/WRM_247_rgb_full.pdf

[36] Yaghmour, K., Masters, M., Ben-Yossef, G., Gerum, P. 2008. Building Embedded Linux Systems. Sebastopol, O'Reilly Media, Inc. 439 s.

LIITE A. LAITEAJURIN RPM FUNKTIOLISTAUS

```
static irqreturn_t irq_tc_a(int irq, void *cookie);
static irqreturn_t irq_tc_b(int irq, void *cookie);

int rpm_ioctl(struct inode *inode, struct file *filep, unsigned int
cmd, unsigned long arg);
int rpm_read(struct file *filp, char *buf, size_t cnt, loff_t *f_pos);
int rpm_open(struct inode *inode, struct file *filp);
int rpm_release(struct inode *inode, struct file *filp);

int init_module(void);
void cleanup_module(void);
```

LIITE B. LAITEAJURIN MPPC OTSIKKOTIEDOSTO

```
#ifndef MPPC_H
#define MPPC_H

#define MPPC_MODULE_NAME        "mppc"
#define MPPC_FILE_NAME          "/dev/mppc"

#define MPPC_MAJOR 0            /* 0 for dynamic */
#define MPPC_MINOR 0

/* ---------------- FLAGS ---------------- */
/* Masks for flag types */
#define MPPC_MASK_COUNT         0x0000FFFF
#define MPPC_MASK_NOTICE        0x00FF0000
#define MPPC_MASK_WARNING       0xFF000000

/* NOTICES, bits 16...23                  */
#define MPPC_FLAG_SIGNAL_STATE  (1 << 16)
#define MPPC_FLAG_LAST_IN_BUF   (1 << 17)

/* WARNINGS, bits 24...31                 */
#define MPPC_FLAG_NO_VALUE      (1 << 24)
#define MPPC_FLAG_BUFFER_FULL   (1 << 25)
#define MPPC_FLAG_HIGH_FREQ     (1 << 26)
/* --------------------------------------- */

#define MPPC_MODE_PULSE_COUNT   0
#define MPPC_MODE_PULSE_WIDTH   1
#define MPPC_MODE_FREQUENCY     2
#define MPPC_NUM_MODES          3

#define MPPC_IOMASK             0xD0
#define MPPC_SET_MODE           _IOW(MPPC_IOMASK, 1, unsigned)
#define MPPC_SET_MULTIPLIER     _IOW(MPPC_IOMASK, 2, unsigned)
#define MPPC_SET_BUFFER_SIZE    _IOW(MPPC_IOMASK, 3, unsigned)
#define MPPC_SET_BUFFER_SIZE_K  _IOW(MPPC_IOMASK, 4, unsigned)
#define MPPC_SET_USE_GPBR       _IOW(MPPC_IOMASK, 5, unsigned)
#define MPPC_SET_EDGE           _IOW(MPPC_IOMASK, 6, unsigned)
#define MPPC_GET_PULSE_COUNT    _IOW(MPPC_IOMASK, 7, unsigned)
#define MPPC GET BUFFER SIZE     IOW(MPPC IOMASK, 8, unsigned)
#define MPPC_GET_TIMER_VALUE    _IOW(MPPC_IOMASK, 9, unsigned)
#define MPPC_GET_SIGNAL_STATE   _IOW(MPPC_IOMASK, 10, unsigned)

#define MPPC_EDGE_NONE          0
#define MPPC EDGE RISING        1
#define MPPC_EDGE_FALLING       2
#define MPPC_EDGE_EACH          3

#define MPPC_MAX_BUFFER_SIZE    1023
#define MPPC_MAX_BUFFER_SIZE_K  64
#define MPPC_MAX_MULTIPLIER     1024

#endif
```

LIITE C. LAITEAJURIN MPPC FUNKTIOLISTAUS

```
/* Module initialization and cleanup */
int init_module(void);
void cleanup_module(void);

/* File operations */
int mppc_ioctl(struct inode *inode, struct file *filp,
               unsigned int cmd, unsigned long arg);
ssize_t mppc_read(struct file *filp, char __user *buf, size_t len,
                  loff_t *f_pos);
ssize_t mppc_write(struct file *filp, const char __user *data,
                   size_t len, loff_t *fpos);
int mppc_open(struct inode *inode, struct file *filp);
int mppc_release(struct inode *inode, struct file *filp);

/* IRQ handlers for timer counter A and B */
irqreturn_t irq_tc_a(int irq, void *cookie);
irqreturn_t irq_tc_b(int irq, void *cookie);

/* Ring buffer operations */
bool buffer_read(measurement_t *measurement);
bool buffer_write(unsigned int value);
bool buffer_init(void);
void buffer_free(void);

/* Read/write configuration in general-purpose backup register */
void config_write_to_gpbr(void);
bool config_read_from_gpbr(void);
```

LIITE D. MPPC-AJURIN TESTIOHJELMAN KOODILISTAUS

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include "mppc.h"

#define DEFAULT_POLL_TIME   1000000u
#define US_IN_MINUTE        60000000u
#define BUFFER_SIZE         65536
#define NOT_SET             0xffffffffu

typedef struct
{
    unsigned int value;
    unsigned int flags;

} measurement_t;

unsigned int avg(measurement_t* buffer, unsigned int size, unsigned int* flags,
                 unsigned int* min, unsigned int* max);

void print_help(void);

int main (int argc, char *argv[])
{
    int fd = 0;
    unsigned int poll_time = DEFAULT_POLL_TIME;
    measurement_t buffer[BUFFER_SIZE];
    unsigned int multiplier = NOT_SET;
    unsigned int buffer_size = NOT_SET;
    unsigned int buffer_size_k = NOT_SET;
    unsigned int mode = NOT_SET;
    unsigned int i = 0;
    unsigned int use_gpbr = NOT_SET;
    unsigned int write_val = NOT_SET;
    unsigned int restore_defaults = 0;
    unsigned int zero = 0;
    unsigned int edge = NOT_SET;
    const char p_zeroes[4] = {0, 0, 0, 0};
    unsigned int read_size = 0;
    unsigned int flags;
    unsigned int read_count;
    unsigned int min;
    unsigned int max;
    unsigned int timer_val;
    unsigned int print_timer = 0;
    unsigned int display_avg = 0;

    if(argc > 1)
    {
        for(i = 1; i < argc; i++)
        {
            if(argv[i][0] == '-')
            {
                switch(argv[i][1])
                {
                case 'a':
                    display_avg = 1;
                    break;
                case 'p':
                    poll_time = atoi(argv[++i]);
                    if(poll_time <= 0 || poll_time > 3600000)
                        poll_time = DEFAULT_POLL_TIME;
                    else
                        poll_time *= 1000;
                    break;
                case 'm':
                    multiplier = atoi(argv[++i]);
                    break;
                case 'b':
                    buffer_size = atoi(argv[++i]);
                    break;
```

```
                case 'B':
                    buffer_size_k = atoi(argv[++i]);
                    break;
                case 'M':
                    mode = atoi(argv[++i]);
                    break;
                case 'g':
                    use_gpbr = atoi(argv[++i]);
                    break;
                case 'e':
                    edge = atoi(argv[++i]);
                    break;
                case 'w':
                    write_val = (unsigned int)atoll(argv[++i]);
                    break;
                case 'd':
                    restore_defaults = 1;
                    break;
                case 'z':
                    zero = 1;
                    break;
                case 't':
                    print_timer = 1;
                    break;
                case 'h':
                    print_help();
                    return 1;
                    break;
                default:
                    printf("Wrong argument %c\n", argv[i][1]);
                    print_help();
                    return 1;
                    break;
                }
            }
        }
    }

    fd = open(MPPC_FILE_NAME, O_RDWR);

    if(fd <= 0)
    {
        perror("Opening device file failed");
        return -1;
    }
    usleep(100000);

    if(write_val != NOT_SET)
    {
        if(write(fd, (char*)&write_val, 4) != 4)
        {
            perror("Write");
            return -1;
        }
        else
            printf("Value %u written\n", write_val);
        close(fd);
        return 0;
    }

    if(restore_defaults)
    {
        ioctl(fd, MPPC_SET_USE_GPBR, 0);
        ioctl(fd, MPPC_SET_MODE, 0);
        ioctl(fd, MPPC_SET_BUFFER_SIZE, 1);
        ioctl(fd, MPPC_SET_MULTIPLIER, 1);
        ioctl(fd, MPPC_SET_EDGE, 1);
    }

    if(use_gpbr != NOT_SET)       ioctl(fd, MPPC_SET_USE_GPBR, use_gpbr);
    if(mode != NOT_SET)           ioctl(fd, MPPC_SET_MODE, mode);
    if(buffer_size != NOT_SET)    ioctl(fd, MPPC_SET_BUFFER_SIZE, buffer_size);
    if(buffer_size_k != NOT_SET) ioctl(fd, MPPC_SET_BUFFER_SIZE_K, buffer_size_k);
    if(multiplier != NOT_SET)     ioctl(fd, MPPC_SET_MULTIPLIER, multiplier);
    if(edge != NOT_SET)           ioctl(fd, MPPC_SET_EDGE, edge);
```

```
    do {
        usleep(poll_time);
        if(mode == MPPC_MODE_PULSE_COUNT)
        {
            read_size = 8;
        }
        else
        {
            read_size = ioctl(fd, MPPC_GET_BUFFER_SIZE) * sizeof(measurement_t);
        }
        read_count = 0;

        do {
            read_count += read(fd, (char*)&buffer[read_count], read_size-read_count);
            usleep(10);
        } while(read_count < read_size);

        if(print_timer)
        {
            timer_val = (unsigned int)ioctl(fd, MPPC_GET_TIMER_VALUE);
            printf("%02u", timer_val / 60000000); //  min
            timer_val = timer_val % 60000000;
            printf(":%02u", timer_val / 1000000); // s
            timer_val = timer_val % 1000000;
            printf(".%03u", timer_val / 1000);    // ms
            printf("\n");
        }

        if(display_avg)
        {
            printf("%6u %9u", read_size/sizeof(measurement_t), avg(buffer, read_count /
                   sizeof(measurement_t), &flags, &min, &max));
            if(flags & MPPC_MASK_WARNING)
                printf(" %x", flags & MPPC_MASK_WARNING);
            printf(" %9u %9u", min, max);
            printf("\n");
        }
        else
        {
            for(i = 0; i < read_size / 8; i++){
                printf("%9u", buffer[i].value);
                if(buffer[i].flags & MPPC_MASK_WARNING)
                    printf(" %x", buffer[i].flags & MPPC_MASK_WARNING);
                printf("\n");
            }

        }

        if(zero)
        {
            if(write(fd, p_zeroes, 4) != 4)
                perror("Write");
        }
    } while(1);


    close(fd);

    return 0;
}
```

LIITE E. DIGITAALISTEN SISÄÄNTULOJEN KYTKENTÄKAAVIO